Machine Learning, 4, 293-336 (1989)
© 1989 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Automated Knowledge Acquisition for Strategic Knowledge

THOMAS R. GRUBER GRUBER@SUMEX-AIM.STANFORD.EDU
Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305

Abstract. *Strategic knowledge* is used by an agent to decide what action to perform next, where actions have consequences external to the agent. This article presents a computer-mediated method for acquiring strategic knowledge. The general knowledge acquisition problem and the special difficulties of acquiring strategic knowledge are analyzed in terms of *representation mismatch*: the difference between the form in which knowledge is available from the world and the form required for knowledge systems. ASK is an interactive knowledge acquisition tool that elicits strategic knowledge from people in the form of *justifications* for action choices and generates *strategy rules* that operationalize and generalize the expert's advice. The basic approach is demonstrated with a human-computer dialog in which ASK acquires strategic knowledge for medical diagnosis and treatment. The rationale for and consequences of specific design decisions in ASK are analyzed, and the scope of applicability and limitations of the approach are assessed. The paper concludes by discussing the contribution of knowledge representation to automated knowledge acquisition.

Key Words. knowledge acquisition, knowledge engineering, human-computer interaction, strategic knowledge, knowledge representation

1. Introduction

Knowledge acquisition is the transfer and transformation of knowledge from the forms in which it is available in the world into forms that can be used by a knowledge system (adapted from [Buchanan, et al. 1983]). In the context of this article, knowledge in the world comes from people and knowledge in the system is implemented with formal symbol structures—knowledge representations. Knowledge acquisition is a multifaceted problem that encompasses many of the technical problems of knowledge engineering, the enterprise of building knowledge systems. Deciding what knowledge can be brought to bear for a problem, how the knowledge can be used by a program, how to represent it, and then eliciting it from people and encoding it in a knowledge base are all aspects of the knowledge acquisition problem. The inherent difficulty of these tasks makes knowledge acquisition a fundamental obstacle to the widespread use of knowledge system technology.

The research reported here addresses the problem of acquiring strategic knowledge from people. In particular, the article presents an approach by which an interactive computer program assists with the knowledge acquisition process. The general term *automated knowledge acquisition* refers to computer-mediated elicitation and encoding of knowledge from people.

The first section of this article provides a theoretical analysis of the general knowledge acquisition problem and introduces the problem of acquiring strategic knowledge. Section 2 reviews the techniques of automated knowledge acquisition in terms of the theoretical framework developed in the first section and motivates the present work. Section 3 describes the automated knowledge acquisition tool called ASK. Section 4 demonstrates the program with a human-computer dialog. Sections 5, 6, and 7 provide an analysis of the scope of applicability, assumptions, and limitations of the system, and a discussion of key design decisions. A concluding section summarizes the contribution of the design of knowledge representations to the development of knowledge acquisition tools.

1.1. The Knowledge Acquisition Problem as Representation Mismatch

Most knowledge systems are built by knowledge engineers rather than by the domain experts who provide the knowledge. A long-standing goal of a course of knowledge acquisition research has been to replace the knowledge engineer with a program that assists in the direct "transfer of expertise" from experts to knowledge bases [Davis 1976]. Yet the problem has eluded a general solution; no existing knowledge acquisition program can build a knowledge system directly from experts' descriptions of what they do.

Why is knowledge acquisition difficult to automate? It seems that the "transfer" metaphor is misleading. Clearly, the form in which knowledge is available from people (e.g., descriptions in natural language) is different from the form in which knowledge is represented in knowledge systems. The difference between the two forms of knowledge, called *representation mismatch* [Buchanan, et al. 1983], is central to the problem of knowledge acquisition. Because of representation mismatch, one cannot merely transfer knowledge from human to machine. The knowledge acquisition tool must actively elicit knowledge in a form that can be obtained from domain experts and map elicited knowledge into the executable representations of the knowledge system. The mapping is difficult to automate because the requirements for building a working system (e.g., operationality, consistency) differ from the requirements for a human expert describing a procedure to another person. In order to automate knowledge acquisition, one must provide a method for overcoming representation mismatch.

The following discussion introduces three aspects of representation mismatch—modeling, operationalization, and generalization—as an explanatory framework with which to understand the problem of knowledge acquisition. The general issues and the specific problems of acquiring strategic knowledge are described within this framework.

1.1.1. Dimensions of Representation Mismatch The *modeling* or formalization problem is a fundamental kind of representation mismatch. A knowledge system can be thought of as a qualitative model of systems in the world, including systems of intelligent activity [Clancey 1989]. While the model embodied by a knowledge system is informed by the behavior of human experts, it is not designed as a model of the experts' knowledge or their cognitive processes [Winograd and Flores 1986]. From this point of view, knowledge acquisition is a creative rather than imitative activity, resulting in a computational model that makes distinctions and abstractions not present in the initial language of the expert. Because of the difference between descriptions of behavior and computational models of action, the task of knowledge acquisition requires a model-building effort beyond that of rendering the expert's utterances in formal notation. Morik [1988] illustrates the modeling problem with the example of building a natural language-understanding system. The builder of such a system does not interview experts in natural language understanding (native speakers) but experts in modeling the formal structure and mechanisms of language (linguists). Furthermore, the system-builder must adapt the expert's concepts (a theory of syntax) to the needs of a computational model (a parser) and sometimes invent new concepts (semantic networks).

The *operationalization* aspect of representation mismatch refers to the difference between descriptions of what the system should accomplish, given by domain experts, and the operational methods for achieving those objectives required by a computer program. Two senses of operationalization have been identified in the machine learning literature: making advice executable [Mostow 1983] or more useful [Keller 1988].[1] Knowledge acquisition involves both kinds of operationalization in the service of performance goals such as recommending an effective drug therapy or designing an efficient electric motor. To make a therapy recommendation executable, a knowledge engineer might build an interface that justifies a recommendation and requests the results. To make the advice "minimize cost, maximize speed" more useful, the engineer might decide to use a redesign algorithm and elicit more knowledge from the expert about ways to cut costs and fine tune performance by modifying existing designs. The methods in which expert-supplied specifications are operationalized may require concepts and terminology unfamiliar to the domain expert.

A third dimension of representation mismatch is *generalization*: the difference between a set of specific examples of desired input/output performance and a more concise representation that will enable a system to perform correctly on a larger class of input situations. It is frequently observed that it is much easier to elicit examples of expert problem solving than general rules or procedures that cover the examples. The available form of knowledge (classified examples) needs to be mapped into a more useful representation (general class descriptions).

Problems of modeling, operationalization, and generalization are ubiquitous in knowledge acquisition. We will now see how they are manifest in the case of a particular kind of knowledge, strategic knowledge.

1.2. The Problem of Acquiring Strategic Knowledge

1.2.1. Strategic Knowledge. *Strategic knowledge* is knowledge used by an agent to decide what action to perform next, where actions can have consequences external to the agent. The more general term *control knowledge* refers to knowledge used to decide what to do next. What constitutes an action and its consequences depends on how one characterizes what the agent can *do*. For knowledge systems that make recommendations to people (e.g., "increase dosage of drug D") or control physical systems (e.g., "close valve V"), actions have consequences that are observable in the world outside of the agent. For problem-solving programs based on state–space search, an action may be the firing of a rule or an operator. For such an agent, *search-control knowledge* is used to choose internal actions that increase the likelihood of reaching a solution state and improve the speed of computation. The research reported here distinguishes knowledge for deciding among actions with consequences in the external world because the goal is to acquire strategic knowledge from domain experts without reference to the symbol-level organization of the knowledge system.

For descriptive purposes, strategic knowledge is also distinguished from the *substantive knowledge* of a domain, knowledge about what is believed to be true in the world. Both substantive and strategic knowledge underlie expertise in many domains. For example, a robot uses substantive knowledge to recognize and interpret situations in the world (e.g., an obstacle in its path) and strategic knowledge to decide what to do (to go around or over it). A lawyer uses substantive knowledge to identify the relevant features of cases and strategic knowledge to decide which case to cite in defense of an argument. A diagnostician uses substantive knowledge to evaluate evidence pro and con hypotheses and uses strategic knowledge to decide among therapeutic actions. In general, substantive knowledge is used to identify relevant states of the world, and strategic knowledge is used to evaluate the utility of possible actions given a state.

1.2.2. Representation Mismatch for Strategic Knowledge Although progress has been made in automating the acquisition of substantive knowledge used in classification [e.g., Bareiss 1989; Boose and Bradshaw 1987; Eshelman 1988], strategic knowledge is typically imparted to systems by knowledge engineers using implementation-level mechanisms. The difficulty of acquiring strategic knowledge directly from experts can be seen within the framework of the three aspects of representation mismatch introduced earlier.

First, strategic knowledge presents serious modeling problems. While substantive knowledge might be acquired in a perspicuous form, such as rules mapping evidence to hypotheses, strategic knowledge about choosing actions is often represented with programming constructs, such as procedures or agenda mechanisms. At least in principle, rules that encode substantive knowledge can be written in a process-independent context; experts can specify how to classify situations in the world without worrying about the mechanism by which the specifications are interpreted. However, specifying knowledge that affects the order and choice of actions involves building a computational model of a process.

Consider the problem of modeling the strategy of a medical *workup*: the process of gathering data, assessing the results, and planning treatment for an individual patient. Although medical diagnosis is often described as a static classification problem (i.e., to classify *given* data), in medical practice evidence for a diagnosis is gathered over time, and the actions that produce evidence are chosen strategically. In modeling the workup, requests for patient data, laboratory tests, diagnostic procedures, and options for trial therapy are treated as actions. Substantive knowledge is used for the classification task, identifying likely causes for a given set of findings. In addition, strategic knowledge is used to decide what action to take next when the data are not all in.

In the MYCIN system, much of the knowledge that determined question ordering and decisions about laboratory tests was represented with screening clauses, clause ordering, and "certainty factor engineering"—implementation-level manipulations of the rules to achieve the intended strategic behavior [Clancey 1983a]. This knowledge could not be acquired easily with the available rule editors and debugging support tools [Buchanan and Shortliffe 1984] because the strategy was implicit in the engineering tricks rather than the content of the rules. Since MYCIN, more explicit representations of strategic knowledge have been devised, such as the control blocks of S.1 [Erman, Scott, and London 1984] and the high-level control languages of BB1 [Hayes-Roth, et al. 1987]. Because these advances are general-purpose languages for control, rendering strategic knowledge in a computational model remains a programming task.

The acquisition of strategic knowledge also highlights the operationalization aspect of representation mismatch. At the *knowledge level* [Newell 1982], the strategic knowledge of an agent may be specified as a set of behavioral goals that the agent should attempt to achieve. While it is possible to elicit specifications of desired behavior at the knowledge level from experts, it is far more difficult for experts (and knowledge engineers) to specify *how* a knowledge system should achieve these goals.

For example, during conventional knowledge acquisition for a knowledge system called MUM [Cohen, et al. 1987], knowledge engineers interviewed a practicing physician for the purpose of modeling his diagnostic strategy for patients reporting chest and abdominal pain. MUM's task was to *generate* workups for chest pain patients, choosing one action at a time, waiting for the outcome of previous action. When asked to describe how to choose diagnostic tests, the expert would mention goals such as "do the cheap, quick tests first" and "protect the patient against a dangerous disease." This is nonoperational advice. To make it operational requires specifying how actions achieve goals (e.g., the diagnostic and therapeutic effect of actions), how to determine the currently relevant goals (e.g., when is a dangerous disease suspected), and how to balance competing objectives (e.g., cost, timeliness, diagnostic power, therapeutic value).

Third, the generalization aspect of representation mismatch is exhibited by the problem of acquiring strategic knowledge. By definition, experts are good at what they do; it does not follow that they are good at generalizing what they do. In particular, it is much easier to elicit *cases* of strategic decisions—choices among actions in specific situations—than to elicit general strategies.

For example, in the MUM domain of chest pain workups, the physician makes a series of decisions about actions. He typically starts with a set of questions about patient history, then performs a physical examination (in a knowledge system, steps in the examination are also implemented as requests for data), and then plans and executes a series of diagnostic tests and trial therapeutic actions, until sufficient evidence for a conclusive diagnosis or recommended therapy has been found. For MUM it was feasible to elicit example workups corresponding to actual patients. These workups can be viewed as very specific plans. Each step in the workup, each choice of what to ask to try next, is the result of a strategic decision. However, generalizations about classes of strategic decisions were not present in the original workup descriptions but developed by retrospective analysis of the cases and follow-up consultation with the expert. Within a single workup there may be several actions chosen for the same reasons (e.g., "do the cheap, quick tests first"), and there may be common reasons across workups (e.g., "gather enough evidence to recommend therapy").

Although cases of specific workups can be acquired in the form of directed graphs, they are not general enough for a knowledge system. First, they are specific to individual patients, and workups differ over individuals. Second, these plan-like procedures are extremely brittle; if any action cannot be taken (e.g., because the results of a test are not available), then the procedures fail. Third, because they only record the results of strategic decisions, workup graphs fail to capture the underlying reasons for selecting actions in the prescribed order. This third problem reveals a subtle form of representation mismatch: although it is possible to elicit reasons for past strategic decisions, these reasons alone do not constitute a *generative* strategy. A generative strategy plans new workups based on the strategic knowledge that gave rise to existing workups.

The work reported in this article is motivated by the problem of acquiring knowledge that underlies strategic decisions and putting it in operational, general form. The next section lays out some of the techiques for addressing the problem.

2. Techniques for Overcoming Representation Mismatch

Interactive tools can assist with knowledge acquisition by overcoming representation mismatch. This section reviews the techniques used by existing knowledge acquisition tools and motivates the approach taken in ASK. The techniques are presented in the context of the three aspects of representation mismatch.

2.1. Incorporating Models into Knowledge Acquisition Tools

Conventionally, the modeling problem for knowledge acquisition is handled by the knowledge engineer, who is responsible for building the knowledge system. The engineer analyzes the *performance task* (the problem to be solved by the knowledge system) and designs a program for applying knowledge to perform the task. A performance task is defined in terms of the input and output requirements of the system and the knowledge that is available. Tasks can be described at multiple levels of abstraction, from the functional specifications for a single application to input/output requirements for a general class of tasks. A *problem solving method* is the technique by which a knowledge system brings specific knowledge to bear on the task. When the computational requirements and methods for a *class* of tasks are well understood, a domain-independent problem-solving method can be designed, such as heuristic classification [Chandrasekaran 1983; Clancey 1985].

A *task-level architecture* consists of a knowledge representation language (a set of representational primitives) and a procedure implementing the problem-solving method designed to support knowledge systems for a class of performance tasks [Chandrasekaran 1986; Gruber and Cohen 1987]. The procedure, which in this article is called the *method* for short, is a mechanism by which knowledge stated in the architecture's knowledge representation is applied to perform one of the tasks in the abstract class of tasks for which the architecture is designed. The representation and the method of a task-level architecture are tightly coupled. Each method defines *roles* for knowledge: ways in which knowledge is applied by the method [McDermott 1988]. The algorithm that implements the method in a program operates on statements in the associated representation language. The primitive terms in the representation correspond to the roles of knowledge. For example, Chandrasekaran [1987] and his colleagues have built architectures for *generic tasks* such as hierarchical classification and routine design. Each generic task is described in terms of the function to be performed (an abstract description of the performance task), a knowledge representation language (the set of primitive terms), and a control strategy (the procedure that implements the method). Chandrasekaran uses the term *generic task problem solvers* to refer to task-level architectures.

Task-level architectures can facilitate knowledge acquisition. Like a virtual machine, the architecture supports a set of method-specific representation primitives for building a knowledge system. Much of the model-building effort can be put into the design of the architecture, and the representational primitives can hide the implementation details. As a consequence, the architecture can reduce representation mismatch by presenting a task-level representation language comprehensible to the domain expert [Bylander and Chandrasekaran 1987; Gruber and Cohen 1987; Musen 1989].

Interactive knowledge acquisition tools can help overcome representation mismatch by employing special techniques for eliciting and analyzing knowledge in architecture-supported representations. Some tools help analyze the task requirements to choose among existing methods and instantiate an architecutre with domain terminology. For example, ROGET [Bennett 1985] offers help in choosing among a small set of particular heuristic classification methods and elicits domain-specific instantiations of the input, output, and intermediate concepts for the selected method.

Other tools specialize in eliciting the knowledge for the roles required by the problem-solving method. For example MOLE [Eshelman 1988] uses an instantiation of the heuristic classification method called cover-and-differentiate. The knowledge acquisition tool specializes in the elicitation of knowledge for roles such as "covering knowledge" and "differentiating knowledge." Similarly, SALT [Marcus 1988] is based on the propose-and-revise method for constructive problem solving, and elicits knowledge for proposing design extensions, identifying constraints, and backtracking from violated constraints.

Tools of another category specialize in a particular formulation of knowledge, independent of how the knowledge will be applied to particular tasks. For example, repertory grid tools elicit knowledge in the form of a two-dimensional matrix of weighted associations between "elements" and "traits" [Boose and Bradshaw 1987; Shaw and Gaines 1987]. These tools use a task-independent elicitation technique to help the user identify traits and elements and the strengths of associations among them and provides detailed analyses of the information. The user interprets the feedback in terms of a particular task, such as a procurement decision or an evaluation of policy alternatives.

On the other end of the spectrum are elicitation tools that are customized to the problem-solving method and a specific task in a domain. An example is OPAL, which acquires protocols used in the domain of cancer therapy [Musen, Fagan, Combs, and Shortliffe 1987]. The problem-solving method is a kind of skeletal-plan refinement, and the performance task is to manage cancer-therapy protocols modeled as skeletal plans. OPAL elicits knowledge from experts entirely in domain-specific terms and in forms that correspond to paper and pencil representations familiar to the experts. Because the tool has almost completely eliminated the representation mismatch due to modeling, it has been used successfully by physicians with little experience with computation [Musen 1989].

The acquisition of strategic knowledge, as it has been defined, is not supported by conventional task-level architectures. In fact, all of the built-in methods of the architectures mentioned above are implemented with procedures that themselves encode a control strategy. To the extent that the strategy is implemented by the method, it cannot be acquired by tools that assume the method is fixed.

However, it is possible to design an architecture for a restricted class of tasks that require domain-specific strategic knowledge. The method for such an architecture should define roles for strategic knowledge, just as MOLE's method defines roles for substantive knowledge, such as knowledge for proposing explanations that cover an abnormal symptom. As will be described in Section 3, ASK was designed with an architecture that represents strategic knowledge as rules that map situations to desired actions. In this architecture, strategic knowledge is limited to three roles for associating features in the agent's current model of the world with classes of appropriate actions. As will be discussed in Section 6, the restricted roles for strategic knowledge reduce the scope of what needs to be acquired and simplify how elicited knowledge is operationalized and generalized. They also limit the class of strategies that can be acquired.

2.2. Eliciting Knowledge in Operational Terms

Automated knowledge acquisition tools can address the operationalization aspect of representation mismatch by limiting what is elicited from the user to representations of knowledge that are already machine-executable—that is, to elicit knowledge in the form in which it will be used for performance or in some form that can be compiled into the runtime representation. An alternative approach is to provide a nonoperational "mediating representation" for eliciting the conceptual structure of a domain and then *manually* building a system that operationalizes the specifications [Johnson and Tomlinson 1988]. A rule editor is a simple example of a tool that elicits knowledge in a form that can be directly executed.

The technique of eliciting knowledge directly in executable form is reminiscent of the single representation trick [Dietterich, et al. 1982] in which the learning agent is given training data in the same representation as the language used for describing learned concepts. Using this technique in a knowledge acquisition tool replaces the problem of making the elicited input executable (operationalization) with the assumption that the elicitation language is representationally adequate. A language is representationally adequate if all of the relevant domain knowledge can be stated in the representation.

The success of tools employing this technique depends in part on whether the elicitation interface can make the operational semantics of the representation comprehensible to the user. For example, although TEIRESIAS paraphrases rules into English, the user needs to know more than English to understand them. TEIRESIAS depends on the assumption that the user can understand the backward-chaining model [Davis 1976].

Well-designed user interface techniques can help make the computational model of the architecture comprehensible to the user. For example, the OPAL tool facilitates the acquisition of cancer treatment protocols with a form-filling interface, emulating paper-and-pencil forms familiar to its users [Musen, et al. 1987]. Similarly, spreadsheet applications are made comprehensible by presenting a familiar metaphor. The interface design goal is to minimize the conceptual distance between the user's understanding of the system's mechanism and the system's presentation of the options afforded by the computational model [Hutchins, Hollan, and Norman 1986].

A tool that acquires knowledge in an executable representation can also offer intelligent assistance by analyzing the consequences of *applying* the knowledge. For example, SALT elicits fine-grained rules for repairing local constraint violations in a design task. One of the consequences of using backtracking from local constraint violations is that the user can unintentionally define cycles in the dependency network, in which repairing one constraint violation introduces another. SALT can analyze the elicited knowledge, identify cycles, and offer assistance to the user in specifying different routes for backtracking [Marcus 1987].

It is difficult to acquire strategic knowledge in executable form without forcing the expert to understand symbol-level mechanisms such as procedures and priority schemes. There is a tension between the requirement to provide the user with a language that is comprehensible and yet sufficiently powerful to implement the strategy. There are some techniques that help elicit specifications of control, such as visual programming interfaces for building transition networks [Musen, Fagan, and Shortliffe 1986] and graph-drawing tools for specifying decision trees [Hannan and Politakis 1986]. However, the strategic knowledge that can generate decisions among actions is *implicit* in transition networks and decision trees.

ASK's representation of strategic knowledge was designed to correspond to the form in which experts can describe their strategic knowledge: justifications for specific actions in specific situations. As will be explained in Section 3, ASK elicits justifications for choices among actions in terms of features of strategic situations and actions. ASK's design ensures that the features mentioned in justifications are operational; the features are well-defined functions and relations that hold over objects in a knowledge base representing the current state of problem solving.

Like all tools that elicit knowledge in executable form, ASK is based on the assumption of representational adequacy discussed above. There are two ways this assumption can fail: the computational model is inadequate for describing the desired strategy, or the set of terms in the existing knowledge representation is incomplete. The former problem is a function of the architecture, as discussed above. The problem of incomplete terms can be handled in an interactive tool if the user is given the chance to define new terms with the representational primitives provided by the architecture.

Since defining terms for a knowledge system is an operationalization task, it is a challenge to provide automated assistance. A promising approach is exemplified by PROTÉGÉ, a tool that helps the knowledge engineer define domain-specific instantiations of architecture-level representational primitives [Musen 1989]. PROTÉGÉ generates OPAL-class elicitation tools meant for the domain expert in which the vocabulary is fixed. ASK provides a means for defining new features in the context of eliciting justifications, as demonstrated in Section 4.6. By design, ASK integrates the acquisition of new features and the acquisition of knowledge that uses the features.

2.3. Integrating Mechanical Generalization with Interactive Knowledge Elicitation

Machine learning techniques are an obvious answer to the generalization aspect of representation mismatch. There are many well-established techniques for generalization from examples [Dietterich and Michalski 1983]. Because inductive generalization is inherently underconstrained, these techniques all depend on some kind of *bias* to direct the learner toward useful or relevant generalizations [Mitchell 1982; Utgoff 1986]. Bias can be provided to a learner by supplying a highly constrained generalization space, defined by the language for representing learned concepts, such as LEX's pattern-matching language [Mitchell, Utgoff, and Banerji 1983]. Bias can also come from the choice of features in the training examples, as in the feature vectors used by decision tree algorithms [Quinlan 1986].

A knowledge acquisition tool can capitalize on existing techniques if they are augmented with the appropriate bias. One approach would be to build the necessary bias into the tool.

If the bias is itself important domain knowledge, however, this approach limits the usefulness of automating the knowledge acquisition process, since the tool would have to be modified for each domain. Instead, a knowledge acquisition tool can provide means for the *user* to contribute bias—to guide the generalization toward useful concepts. The user can contribute bias by carefully selecting training examples [Winston 1985], by identifying their relevant features, and by evaluating machine-generated generalizations. While the human provides pedagogical input and evaluation of results, the tool can apply syntactic generalization operators and check for consistency with a database of training cases. The resulting human-machine synergy is a more powerful acquisition technique than either manual knowledge engineering or traditional inductive learning.

Knowledge-based learning techniques such as explanation-based learning [DeJong and Mooney 1986; Mitchell, Keller, and Kedar-Cabelli 1986] are strongly biased by the domain theory provided by the system builder. Inserting a human in the learning loop can help overcome the dependence of the learning technique on the quality of the built-in knowledge. For example, in an experiment with SOAR in the domain of algebraic simplification, a human intercedes during problem solving to help the system learn search-control knowledge [Golding, Rosenbloom, and Laird 1987]. When the system needs to choose among algebraic simplification operators for a specific equation, the human recommends an operator to apply or provides a simpler equation to solve. The system uses a domain theory of algebraic simplification to find useful chunks that generalize the situation (the class of equations) in which the recommended operator should be applied. In the absence of a complete domain theory, one can imagine the human pointing out relevant parts of the equation to chunk.

To integrate generalization techniques into a knowledge acquisition tool, the knowledge to be acquired must be represented in such a way that syntactic generalizations of statements in the representation correspond to semantic generalizations in the knowledge [see Lenat and Brown 1984]. For strategic knowledge, this means formulating the selection of actions in terms of classification. For example, a common technique for programs that learn search-control knowledge is to formulate the knowledge for selecting actions as pattern-matching expressions that identify situations in which operators would be usefully applied [Benjamin 1987; Laird, Newell, and Rosenbloom 1987; Minton and Carbonell 1987; Mitchell, Utgoff, and Banerji 1983; Silver 1986]. Because of this formulation, syntactic generalizations of the expressions to which an operator had been applied during training correspond to classes of situations where the operator might be useful in the future.

ASK's representation of strategic knowledge is designed to exploit syntactic generalization operators. Knowledge about what action to do next is formulated as predicates that describe situations in which equivalence classes of actions are useful. In the absence of a theory to infer the utility of actions, ASK acquires strategic knowledge from people.

3. The ASK Knowledge Acquisition Assistant

ASK is an interactive knowledge acquisition assistant. It acquires strategic knowledge from the user of a knowledge system, called the performance system. The strategic knowledge acquired by ASK is used by the performance system to decide what action to perform on each iteration of a control cycle. With additional strategic knowledge, the performance system should be able to make better decisions about what to do in various situations.

The basic approach is to elicit strategic knowledge from the user in the form of *justifications* for specific choices among actions, and then operationalize and generalize the justified choices in the form of *strategy rules* that associate situations with classes of appropriate actions.

This section presents an overview of the knowledge acquisition procedure, and then covers in more detail the strategy-rule representation and the knowledge system architecture that supports it. Section 4 demonstrates ASK with examples from a knowledge system for planning workups of chest pain.

3.1. The Knowledge Acquisition Dialog

ASK orchestrates a mixed-initiative dialog with the user. The basic steps in the knowledge acquisition dialog are shown in Figure 1.

ASK is invoked by the user of the performance system. At run time, the performance system executes a simple control loop. On each iteration the system selects a set of recommended actions, the user picks one, and then the system executes it. The results of the actions are recorded, and then the system continues by selecting the next set of recommended actions. If the user disagrees with the system's recommended actions on any iteration, she can interrupt the control loop and initiate a knowledge acquisition dialog.

The first step of the knowledge acquisition dialog is to elicit a critique from the user. A critique is a labeling of what the system did wrong in terms of choosing actions. The system recommends a set of actions at each iteration of the control cycle because they are all equally appropriate in the current situation, according to the existing strategic knowledge. The user

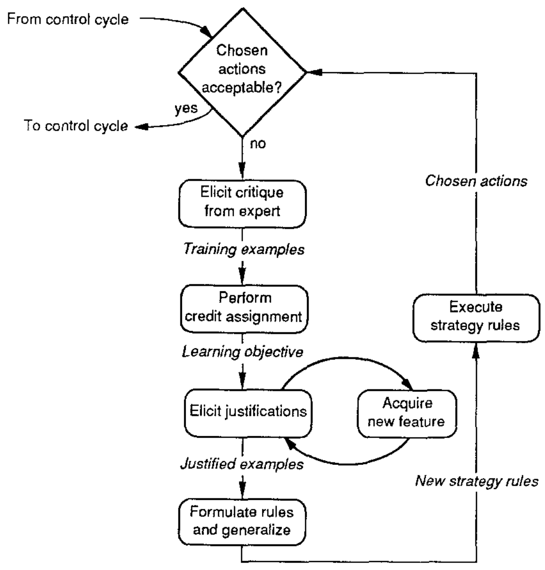


Figure 1. The ASK knowledge acquisition dialog.

critiques the system's choices by selecting an action that the system should have chosen (the positive example) and one that the system should not have chosen (the negative example). The positive and negative examples do not have to be in the set of the system's initial choices (which may be empty). The user also characterizes the system's error in recommending actions, indicating, for instance, whether the positive example is merely preferred to the negative example or whether the negative example should not have been considered at all.

Next, ASK performs credit assignment analysis by examining how the current set of strategy rules matched the positive and negative examples. The output of this analysis is a learning objective that specifies what a new strategy rule would have to match and not match and what it should recommend in order to accommodate the user's critique and be consistent with existing strategy rules.

Then ASK elicits justifications from the user. From the user's perspective, justifications are explanations or reasons why an action should or should not be recommended, in terms of relevant features of the current situation. From ASK's perspective, justifications are facts about the state of knowledge base objects in the current working memory of the performance system; the set of justifications corresponds to the set of features that should be mentioned in matching strategy rules. ASK suggests an initial seed set of justifications, based on how existing strategy rules fired. The user adds justifications by clicking on features of objects displayed in windows on the screen. The justification interface allows the user to browse through the knowledge base for relevant objects. If the set of existing features is inadequate, the user can define new features within the justification interface.

When the user indicates that she is finished and has specified a set of justifications that are sufficient to distinguish the positive and negative examples, ASK generates a new strategy rule from the justifications. The new strategy rule is generalized by syntactic induction operators to apply to a range of situations and an equivalence class of actions. For example, where a specific action appears in a justification, ASK puts a variable in the corresponding clause of a strategy rule. Similarly, if a justification mentions a specific value for a feature, ASK may build a strategy-rule clause that matches a *range* of values for that feature.

Finally the new rule is paraphrased and the operational effects of the new rule are presented to the user for approval. If the user agrees that the new rule improves the system's choices of actions, the rule is added to the strategic knowledge base of the performance system, and the control cycle is continued.

Details of the knowledge acquisition dialog are demonstrated with examples in Section 4. First some background on the performance system architecture and the representation for strategic knowledge is required.

3.2. The MU Architecture

ASK is integrated with an architecture for knowledge systems called MU [Cohen, Greenberg, and Delisio 1987; Gruber and Cohen 1987]. As depicted in Figure 2, a performance system built in MU consists of a substantive knowledge base, typically for heuristic classification, and a strategic knowledge base for controlling actions.[2] This division of knowledge is typical of architectures that support control knowledge, such as BB1 [Hayes-Roth 1985]. MU organizes the substantive knowledge as a symbolic inference network, where inferences are

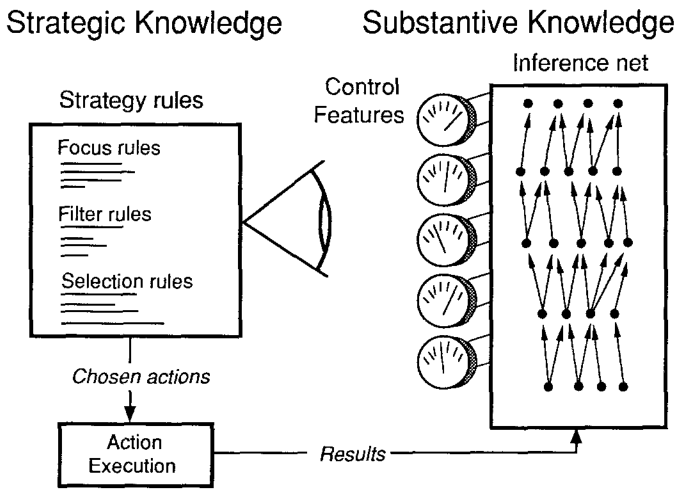


Figure 2. The MU architecture with strategy rules.

propagated from evidence to hypotheses by local combination functions. The inference network serves as the working memory of the system at runtime. The state of the network is abstracted by *control features*, which are functions, attributes, and relations over knowledge base objects.[2] The strategic knowledge is organized in a separate component, which examines the state of working memory via control features and selects actions to execute. MU was designed to support a variety of experiments in strategic reasoning, so the architecture does not include a built-in problem solving method or control strategy. The strategy-rule representation was developed for the study of knowledge acquisition in ASK.

3.3. Strategy Rules

Strategic knowledge acquired by ASK is represented in the form of *strategy rules*, inspired by the metarules that represent diagnostic strategy in NEOMYCIN and HERACLES [Clancey 1988; Clancey and Bock 1988]. Strategy rules map *strategic situations* to sets of recommended actions. Strategic situations are states of the working memory of a performance system. In the MU architecture, strategic situations are states of the inference network.

The strategy-rule control cycle, shown in Figure 3, specifies how strategy rules are applied in a performance system to decide among actions. At each iteration of the control cycle, strategy rules recommend the actions that are appropriate to perform next. There are three types of recommendations, corresponding to three categories of strategy rules. *Focus rules* propose a set of possible actions at each iteration. *Filter rules* prune actions that violate constraints. *Selection rules* pick out subsets of the proposed and unpruned actions that are most desirable in the current situation to form the final set of recommended actions. One of the actions in the recommended set is chosen by the user and executed. The effects of executing the action are then propagated through working memory.

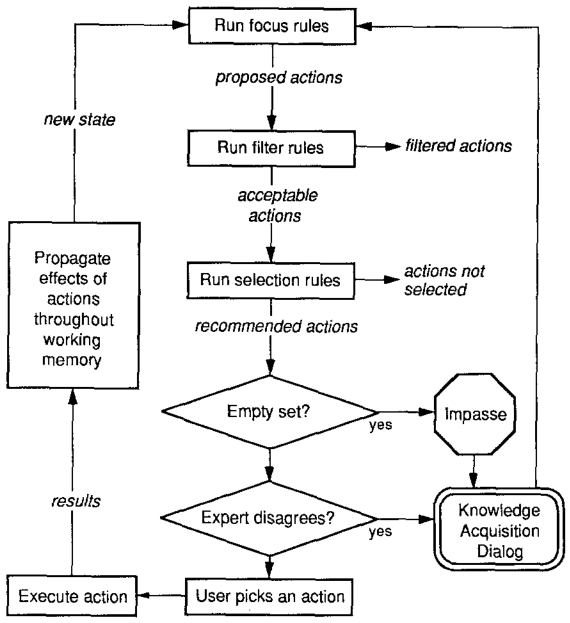


Figure 3. The strategy rule control cycle.

The strategy-rule control cycle corresponds to the method of task-level architectures described in Section 2.1. It specifies how strategic knowledge is brought to bear in the decision about what action to do next. The propose-filter-select algorithm defines three roles for strategic knowledge: specifying the conditions under which actions might be applicable, inappropriate, and preferable. Its design stipulates that actions are chosen iteratively, waiting for the effects of the execution of the previous action before making the current decision. The algorithm also assumes that the context of the decision, the *strategic situtation*, is defined in terms of currently available features of the state of the performance system. Thus, strategy rules are *not* general-purpose control rules, useful for writing arbitrary programs. Rather, the strategy-rule control cycle supports a style of reasoning that has been called *reactive planning* [Agre and Chapman 1987; Chapman and Agre 1987; Firby 1987; Kaelbling 1987]. The form of strategic knowledge is restricted to facilitate automated knowledge acquisition. The consequences of this design are made explicit in later sections.

The left-hand side (If part) of a strategy rule is a conjunctive expression, with variables, that specifies a strategic situation and the set of recommended actions for that situation. The left-hand side expression matches against the values of control features that reflect the properties and dynamic state of objects in working memory, including objects that represent actions. The right-hand side (Then part) of a strategy rule indicates whether the matching actions should be proposed, filtered, or selected in the matching situation.

3.4. Examples from the Chest Pain Domain

Here are some examples of strategy rules and control features from a system for planning workups for chest pain that will be used to demonstrate ASK in Section 4.

The following *focus rule* proposes actions that are general questions (e.g., age, sex, etc.) when the set of active hypotheses, called the differential, is empty.

```
Rule Ask-intake-questions  a focus rule
  ''Ask general questions when at a loss.''
If: (IS (differential) :EMPTY)
    (IN ?ACTION (members-of general-questions))
Then: (PROPOSE ?action history-and-exam)
```

The strategic situation in this rule is specified by the condition that the value of the `differential` object is empty. The set of recommended actions is generated by the relation `members-of` applied to the object `general-questions`, which is a class of actions. The right-hand side operator `PROPOSE` specifies that the values bound to the variable `?ACTION` should be proposed under these conditions, and that the goal `history-and-exam` should be posted.

The expression `(differential)` refers to the set of hypotheses on the differential. It is a control feature defined in the MU inference network as:

```
VALUE of DIFFERENTIAL           a control feature
  ''The set of active hypotheses''
SET-OF ?Hypothesis IN hypotheses SUCH-THAT
  trigger-level OF ?Hypothesis IS triggered AND
  level-of-support OF ?Hypothesis IS-NOT disconfirmed
 OR
  level-of-support OF ?Hypothesis IS-AT-LEAST supported
```

Another focus rule, shown below, is complementary to `Ask-intake-questions`. It proposes actions that potentially provide diagnostic evidence when the differential is *not* empty, and labels this state with the goal `gather-evidence-for-differential`.

```
Propose-diagnostic-evidence          a focus rule
  ''Gather evidence for current hypotheses.''
If: (IS (differential) :NONEMPTY)
    (IN ?ACTION (potential-evidence differential))
Then: (PROPOSE ?ACTION gather-evidence-for-differential)
```

The expression `(potential-evidence differential)` refers to a control feature that returns the set of actions that are potentially diagnostic for hypotheses on the differential. This set is computed dynamically by a function that calls a MU service for analyzing the inference network [Cohen, Greenberg and Delisio 1987].

A very simple *filter rule* prevents actions from being recommended if they have already been executed. In some domains actions may be executed repeatedly. That is why the don't-repeat policy is encoded in the following rule instead of built in to the basic control loop.

```
Filter-executed-actions          a filter rule
   ''Do not repeat actions''
If: (IS (executed? ?ACTION) yes)
Then: (FILTER ?ACTION)
```

The following *selection rule* is enabled under the goal history-and-exam. It recommends those actions that are cheap to perform and that can potentially produce data that would trigger new hypotheses.

```
Select-cheap-triggering-data          a selection rule
   ''Prefer cheap actions that might trigger hypotheses.''
If: (IN history-and-exam (current-goals))
     (IS (potentially-triggered-by ?ACTION) :NONEMPTY)
     (≤ (cost ?ACTION) cheap)
Then: (SELECT ?ACTION)
Shadows: Select-triggering-data, select-free-evidence,
          select-cheap-evidence
```

The terms current-goals, potentially-triggered-by, and cost refer to control features. The set of actions recommended by this rule are those with some hypotheses on their potentially-triggered-by feature and whose cost feature is not more than cheap. The feature potentially-triggered-by is computed from the definitions of triggering conditions for hypotheses, stated in a rule-like form. For example, the hypothesis classic-angina is triggered when "the chief-complaint is pain or pressure and pain-quality is vise-like and the chief-complaint-location is substernal." This rule will recommend the action of asking for the chief-complaint-location because it potentially triggers a hypothesis and it is cheap.

3.5. The Shadowing Relation Among Strategy Rules

Within each strategy-rule category (focus, filter, selection), rules are matched in an order specified by a precedence relation called *shadows*, which is a partial order based on the generality of left-hand sides. If a rule succeeds (matches some objects), then the more general rules that it shadows are pruned (prevented from being fired). Generality is defined in terms of the features mentioned in a rule and the range of values specified for each feature. For example, the selection rule shown above, Select-cheap-triggering-data, shadows (takes precedence over) more general rules mentioning the same features. It shadows the more general rule Select-cheap-evidence, which recommends any action that is cheap. In turn, Select-cheap-evidence shadows the rule Select-free-evidence because the former matches actions with costs of cheap *or* free. The global effect of a family of selection rules in which the more specific rules shadow the more general is to choose those actions judged to be acceptable by the most constraining criteria. The shadows relation is a symbolic alternative to a numeric function for combining the recommendations of each rule into a single measure of utility. Further details can be found in [Gruber 1989].

4. A Knowledge Acquisition Dialog with ASK

In this section, ASK will be demonstrated in the context of a performance system that generates diagnostic workups for patients reporting chest and abdominal pain. The performance system is a reimplementation of the MUM knowledge system [Cohen, et al. 1987]. MUM's task is called *prospective diagnosis*, which is to choose diagnostic actions as a physician would, asking questions in an intelligent order and balancing the potential costs of diagnostic tests and trial therapy with the evidential and therapeutic benefits.

4.1. What the Performance System Already Knows

In experiments with ASK, the performance system is given MUM's substantive knowledge about the diagnosis of chest pain, implemented in the MU architecture in an inference network. The inference network contains hypotheses, data-gathering actions, intermediate conclusions, and combination functions that represent inferential relations such as the evidential support for hypotheses given patient data. MUM's original strategy was written by knowledge engineers in Lisp. In the ASK experiments, the strategic knowledge is represented in strategy rules.

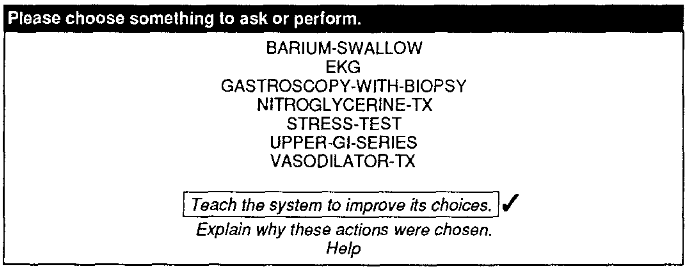
In the dialog shown here, the performance system starts with a small but incomplete set of strategy rules, and the user extends them to improve strategic performance. ASK can also be used without any existing strategy rules. In a separate experiment reported in [Gruber 1989], ASK was used to acquire a set of strategy rules that replicates the original MUM strategy. However, since ASK makes use of existing strategy rules and control features in acquiring new strategic knowledge, it can be more helpful in specializing an existing strategy than in building a strategy from scratch. Thus the dialog in this section will show ASK being used to extend an existing set of rules that represent a general strategy for prospective diagnosis.

4.2. Running the Performance System

A MU performance system runs the basic control loop that was introduced in Section 3.3. At each iteration, strategy rules recommend some set of actions as candidates. From the system's point of view, these recommended actions are equivalent. Given the current strategic knowledge, the system could select among them arbitrarily. The user of a MU system is given the choice to "break the tie" and pick one action to execute. In the chest pain application, executing an action typically causes a request for data (e.g., symptoms or test results). That data is entered into the inference network, where it may change the evidential support for active hypotheses and trigger new hypotheses.

We begin the knowledge acquisition dialog at a point at which the user has already run the performance system through the first several actions in a case (namely, the cheap and easy questions about the history and the physical examination data). At this point, the system has run out of cheap actions and the Propose-diagnostic-evidence rule (Section 3.4) recommends a set of diagnostic actions. The user has the option to pick one of the recommended actions for execution or to teach the system to refine its strategy.

The following menu shows the system offering a set of recommended actions during an iteration of the control cycle of the performance system. Instead of choosing an action, the user initiates the dialog with ASK to "teach the system to improve its choices." (An item with a box drawn around it signifies that the user has selected it with the mouse.) The user sets up this diagnostic situation because it demonstrates a weakness in the system's strategy. The system needs to be more selective in choosing among diagnostic tests and trial therapeutic actions such as the seven offered in the menu.



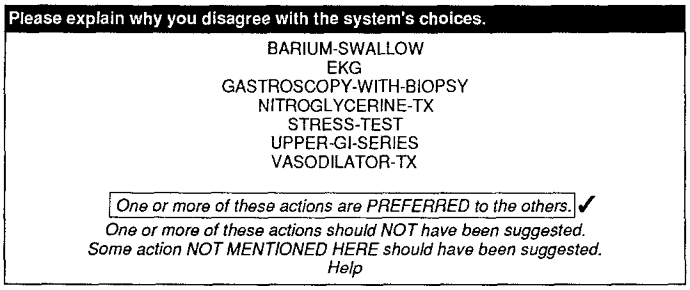
Please choose something to ask or perform.

BARIUM-SWALLOW
EKG
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

[✓ *Teach the system to improve its choices.*]
Explain why these actions were chosen.
Help

4.3. *Eliciting the User's Critique*

ASK elicits a critique from the user by presenting the list of the system's chosen actions and asking what should have been done differently. It first asks for the general category of error, to help determine whether the problem is with focus, filter, or selection rules:

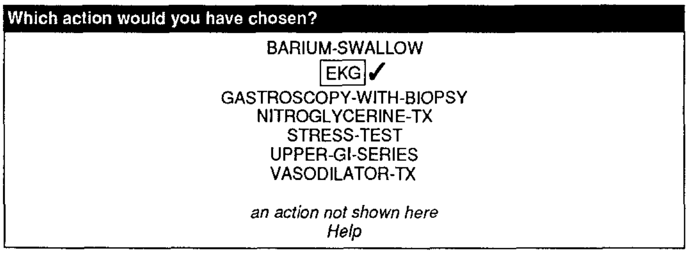


Please explain why you disagree with the system's choices.

BARIUM-SWALLOW
EKG
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

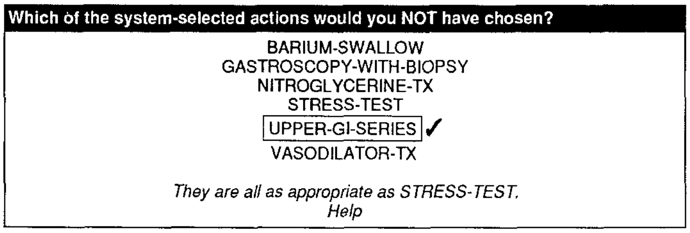
[✓ *One or more of these actions are PREFERRED to the others.*]
One or more of these actions should NOT have been suggested.
Some action NOT MENTIONED HERE should have been suggested.
Help

Then it asks for a positive example, an action that should have been recommended, and a negative example, an action that should not have been recommended. It is assumed that the user will choose a positive example that is representative of a class of actions that should be recommended in this situation, and a negative example that represents a class of actions to distinguish in this situation. In the interaction shown below, the user indicates that the action EKG should have been distinguished from the action Upper-GI-series, which is a reasonable alternative (i.e., a near miss).


Which action would you have chosen?

BARIUM-SWALLOW
[✓ EKG]
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
UPPER-GI-SERIES
VASODILATOR-TX

an action not shown here
Help


Which of the system-selected actions would you NOT have chosen?

BARIUM-SWALLOW
GASTROSCOPY-WITH-BIOPSY
NITROGLYCERINE-TX
STRESS-TEST
[✓ UPPER-GI-SERIES]
VASODILATOR-TX

They are all as appropriate as STRESS-TEST.
Help

4.4. Credit Assignment Analysis

Using the information provided by the user, ASK performs a credit assignment analysis. The credit assignment algorithm examines how existing strategy rules matched in this situation and determines the requirements for a new rule that would account for the critique. The algorithm makes strong use of the distinction between focus, filter, and selection rules and the way they are applied in the strategy-rule control cycle. For example, if the positive example was not proposed by any focus rules, the algorithm prescribes learning a focus rule that proposes it. Alternatively, if both the positive and negative examples are recommended by selection rules, then the algorithm prescribes learning a selection rule that matches the positive example, fails to match the negative example, and shadows the selection rules that recommended the negative example. In the sample session, ASK determines that it needs to acquire a selection rule, specializing the Propose-diagnostic-evidence rule, such that the new rule matches EKG and does not match Upper-GI-series. The complete credit assignment algorithm can be found in [Gruber 1989].

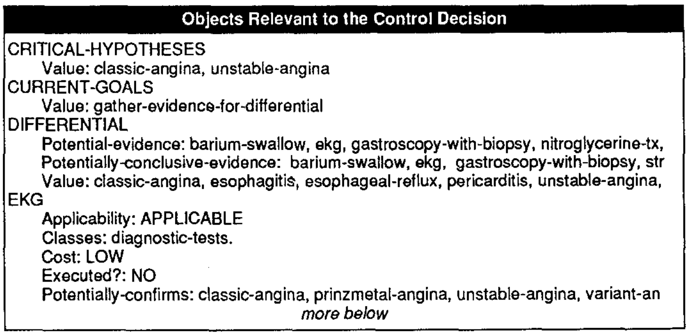
4.5. Eliciting Justifications

In the next stage of the dialog, the user provides justifications for choosing the positive example over the negative example. Justifications are specified as features of the current strategic situation and features of actions. In the example session, the strategic situation is characterized by the state of hypotheses on the differential. A feature shared by the actions recommended by the system (including the positive and negative examples) is that they potentially provide evidence for hypotheses on the differential. In the current example, the user must provide additional justifications that distinguish the positive example EKG from the negative example Upper-GI-series.

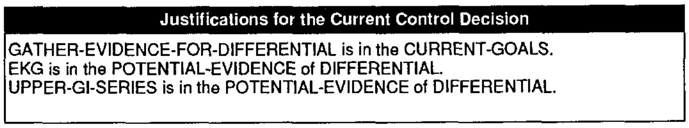
The user interface for asserting justifications consists of two windows containing mouse-sensitive text. The "relevant objects window" displays the values of features of a set of objects from the knowledge base. The "justifications window" contains a list of justifications in the form of natural language sentences. Each justification is a description of the value of a feature of some relevant object.

ASK initializes the relevant objects window with a set of knowledge base objects that might be relevant to the current control decision. An object is considered relevant if it is one of the positive or negative examples (actions), a current goal, an instance of a class representing some aspect of the global state of the inference network, or if it is mentioned in a strategy rule matching the positive or negative examples. The user is provided with tools for browsing the knowledge base to find additional relevant objects.

ASK also initializes the list of statements in the justification window with *seed justifications* which represent the system's reasons for selecting the current actions. Seed justifications are derived from the clauses of strategy rules matching the positive and negative examples. In the windows shown below, objects and justifications have been seeded by ASK.

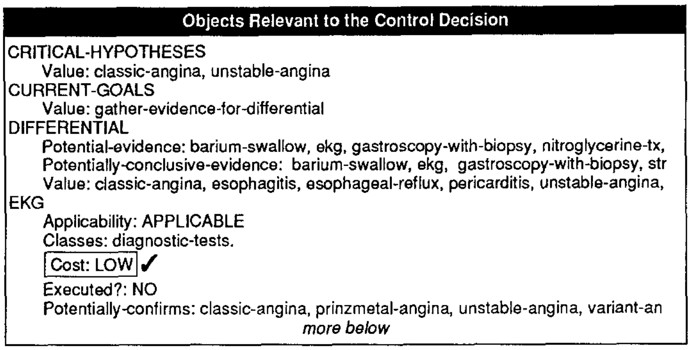


```
Objects Relevant to the Control Decision
CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina
CURRENT-GOALS
    Value: gather-evidence-for-differential
DIFFERENTIAL
    Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
    Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
    Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
    Applicability: APPLICABLE
    Classes: diagnostic-tests.
    Cost: LOW
    Executed?: NO
    Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
                              more below
```



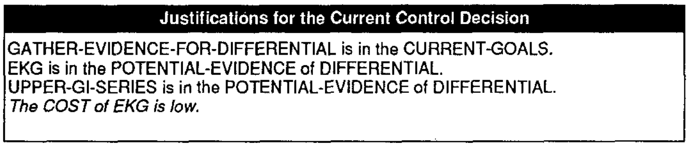
```
Justifications for the Current Control Decision
GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
```

The user asserts a justification by selecting a feature of one of the objects presented in the relevant objects window. When a justification is selected, ASK paraphrases the fact in the justifications window. In the following interaction, the user indicates that EKG should have been chosen because it has low cost. Using the mouse, the user selects the statement "Cost: low" from the relevant objects window, and the statement "The COST of EKG is low" shows up in the justification window, as depicted below.



Objects Relevant to the Control Decision

CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina
CURRENT-GOALS
    Value: gather-evidence-for-differential
DIFFERENTIAL
    Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
    Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
    Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
    Applicability: APPLICABLE
    Classes: diagnostic-tests.
    [✓ Cost: LOW]
    Executed?: NO
    Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
more below



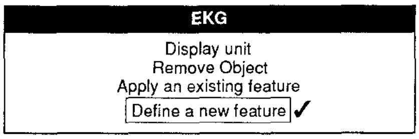
Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of EKG is low.

At this point the user could tell ASK that she was finished. If the set of justifications satisfied the learning objective, ASK would then turn the justifications into a new strategy rule. In this session, however, the user wishes to add more justifications. In particular, the user wants to say that EKG is appropriate in this situation not only because it has low cost, but also because it takes little time to perform. To be able to say this in the language of justifications, the user needs to define a new feature.

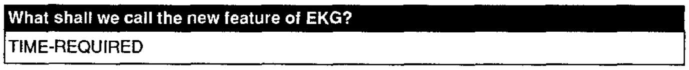
4.6. *Acquiring a New Feature*

To define a new feature is to implement an attribute, function, or relation over some set of objects in the knowledge base. ASK can help the user define a new feature. Playing the role of a knowledge engineer, ASK elicits the information needed to implement the feature in the MU architecture. The interaction below shows the user defining a new feature called "time required." The user starts by clicking on the EKG object in the relevant objects window, bringing up the following menu:



EKG

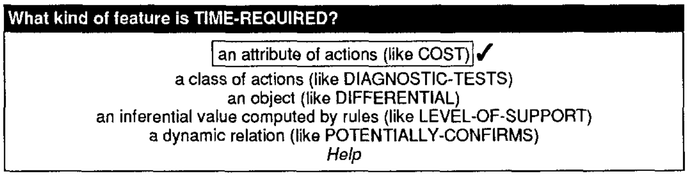
Display unit
Remove Object
Apply an existing feature
[✓ Define a new feature]



What shall we call the new feature of EKG?

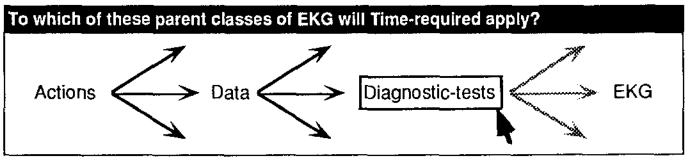
TIME-REQUIRED

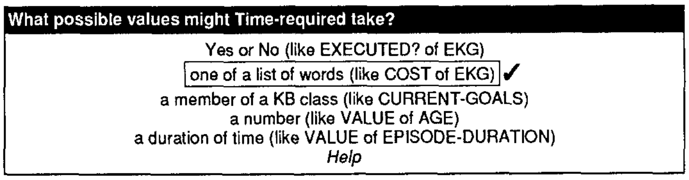
After obtaining a name for the feature, ASK needs to determine its general type. The type of a feature is a symbol-level property, dependent on the knowledge-base architecture. MU supports several varieties of control features, many of which are best implemented by knowledge engineers (e.g., dynamic relations written in Lisp). ASK knows about how features are implemented in MU and makes it possible to acquire some of the more simple features, such as static attributes, interactively. To help make architecture-dependent terms such as "inferential value" concrete to the user, ASK offers instances of features types from the current knowledge base as exemplars. In the menu below, the user indicates that the time-required feature is an attribute of actions, analogous to the cost feature.


What kind of feature is TIME-REQUIRED?

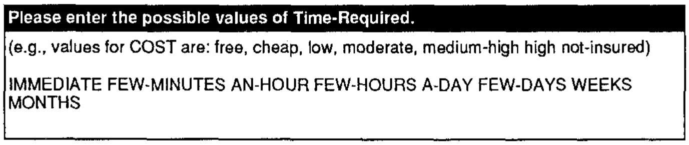
[✓ an attribute of actions (like COST)]
a class of actions (like DIAGNOSTIC-TESTS)
an object (like DIFFERENTIAL)
an inferential value computed by rules (like LEVEL-OF-SUPPORT)
a dynamic relation (like POTENTIALLY-CONFIRMS)
Help

To complete the definition of a static attribute, ASK elicits information about the domain, data type, possible values, order, cardinality, and default value for the feature, and constrains the user's choices whenever possible.


To which of these parent classes of EKG will Time-required apply?

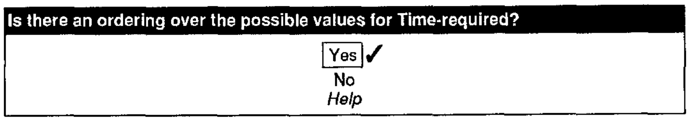

What possible values might Time-required take?

Yes or No (like EXECUTED? of EKG)
[✓ one of a list of words (like COST of EKG)]
a member of a KB class (like CURRENT-GOALS)
a number (like VALUE of AGE)
a duration of time (like VALUE of EPISODE-DURATION)
Help

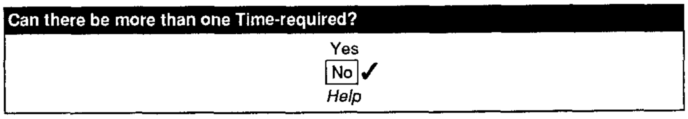

Please enter the possible values of Time-Required.

(e.g., values for COST are: free, cheap, low, moderate, medium-high high not-insured)

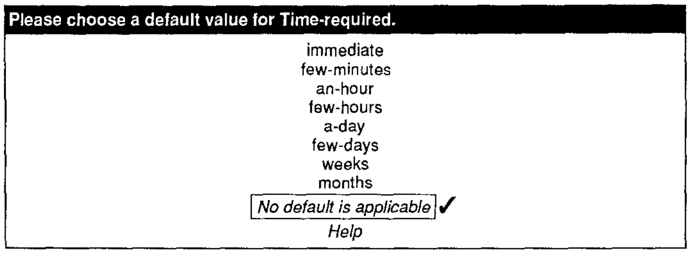
IMMEDIATE FEW-MINUTES AN-HOUR FEW-HOURS A-DAY FEW-DAYS WEEKS MONTHS


Is there an ordering over the possible values for Time-required?

[✓ Yes]
No
Help


Can there be more than one Time-required?

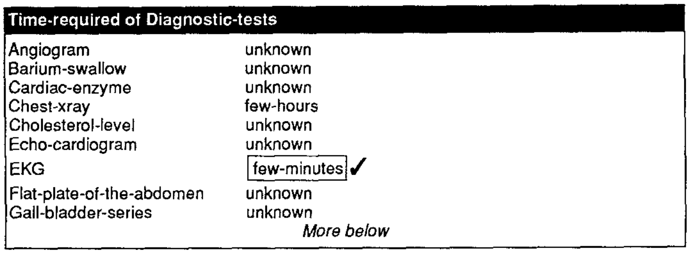
Yes
[✓ No]
Help



Please choose a default value for Time-required.

immediate
few-minutes
an-hour
few-hours
a-day
few-days
weeks
months
[✓ No default is applicable]
Help

Once the intentional properties of the feature are acquired, the values of the feature applied to the elements of its domain are elicited. For static attributes, ASK presents a table of the objects to which it applies, and the user specifies the value of the feature for each object. In the current example, the user enters the value of the time-required feature for all diagnostic tests, including the training examples EKG and Upper-GI-series. The table below shows the value of time-required for EKG, after it was entered by the user.



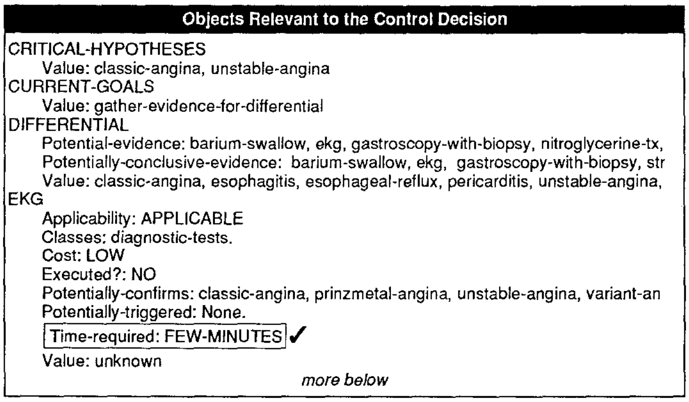
Time-required of Diagnostic-tests

Angiogram	unknown
Barium-swallow	unknown
Cardiac-enzyme	unknown
Chest-xray	few-hours
Cholesterol-level	unknown
Echo-cardiogram	unknown
EKG	[✓ few-minutes]
Flat-plate-of-the-abdomen	unknown
Gall-bladder-series	unknown

More below

4.7. Using the New Feature in Justifications

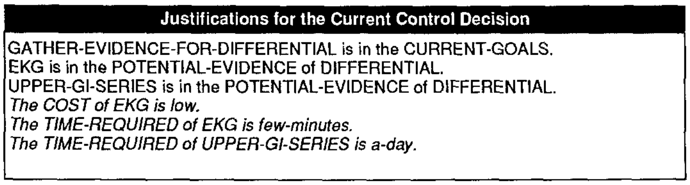
When the expert has finished defining time-required, the system can use it as any other feature and ASK can offer it as a possible justification. The dialog now returns to the justification interface, where the user selects the time-required as a justification for choosing EKG over Upper-GI-series:



Objects Relevant to the Control Decision

```
CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina
CURRENT-GOALS
    Value: gather-evidence-for-differential
DIFFERENTIAL
    Potential-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, nitroglycerine-tx,
    Potentially-conclusive-evidence: barium-swallow, ekg, gastroscopy-with-biopsy, str
    Value: classic-angina, esophagitis, esophageal-reflux, pericarditis, unstable-angina,
EKG
    Applicability: APPLICABLE
    Classes: diagnostic-tests.
    Cost: LOW
    Executed?: NO
    Potentially-confirms: classic-angina, prinzmetal-angina, unstable-angina, variant-an
    Potentially-triggered: None.
    [✓ Time-required: FEW-MINUTES]
    Value: unknown
```

more below



Justifications for the Current Control Decision

GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
EKG is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
UPPER-GI-SERIES is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of EKG is low.
The TIME-REQUIRED of EKG is few-minutes.
The TIME-REQUIRED of UPPER-GI-SERIES is a-day.

At this point in the dialog, the user has indicated that the cost and time required of actions are factors to consider when choosing actions. The first three justifications represent the factors that the system would consider and were suggested by ASK. The user could have removed some of these seed justifications but did not in this case. From the combined set of justifications, ASK can generate a new strategy rule.

4.8. Generating and Generalizing a Strategy Rule

Given the user's justifications, ASK formulates a new strategy rule that accounts for the expert's critique of the system's performance. The new rule causes the expert's preferred action to be selected on the next iteration.

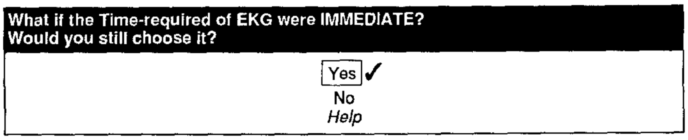
The left-hand side of the new rule is constructed by transforming the list of justifications into left-hand-side clauses. The transformation from justifications to rule clauses is fairly straightforward. The internal representation of justifications is very similar to the clause form of strategy rules. The right-hand-side recommendation (in this case, SELECT) was decided by the credit assignment analysis. In the current example, ASK forms the following rule:

```
IF   (IN gather-evidence-for-differential (current-goals))
     (IN ?ACTION (potential-evidence differential))
     (≤ (cost ?ACTION) low)
     (= (time-required ?ACTION) few-minutes)
THEN (SELECT ?ACTION)
```

In the process of forming rule clauses from justifications, ASK applies generalization operators. One operator is called *turning constants into variables*. In the strategy rule above, references to EKG have been replaced with the free variable ?ACTION, which is bound at runtime by the strategy-rule interpreter to each action that has been proposed and has not been filtered. The result is that the rule recommends the class of actions sharing the features of EKG in the justifications: the cost and time required.

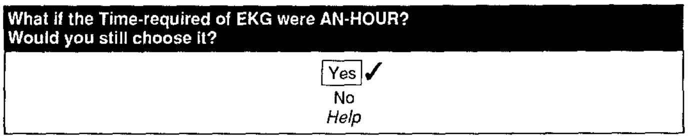
Another generalization operator is *extending the reference* of a feature from a test of equality to a test over some range or set of permissible values. In the example strategy rule, the ≤ operator specifies that the third clause will succeed when the action has any value of cost equal to or less than low. ASK used a heuristic for applying this generalization; it found another selection rule that used ≤ for the cost feature.

In this example, however, ASK has no a priori information to help in extending the reference of the new feature, time-required. It asks the user for guidance by posing hypothetical variants on the current case to obtain boundary conditions on the acceptable range for the time-required clause in this rule. Since ASK lacks common sense, it has to ask whether the user would still accept the EKG if it takes no time at all:

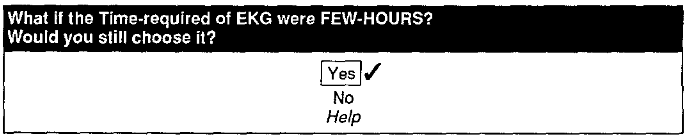

What if the Time-required of EKG were IMMEDIATE?
Would you still choose it?

[✓ Yes]
No
Help

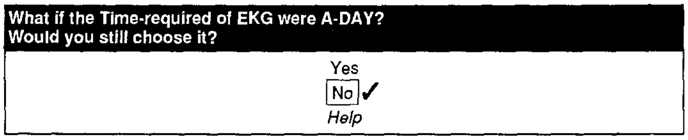
Then ASK offers near-miss cases:


What if the Time-required of EKG were AN-HOUR?
Would you still choose it?

[✓ Yes]
No
Help


What if the Time-required of EKG were FEW-HOURS?
Would you still choose it?

[✓ Yes]
No
Help


What if the Time-required of EKG were A-DAY?
Would you still choose it?

Yes
[✓ No]
Help

Given this information, ASK replaces the clause

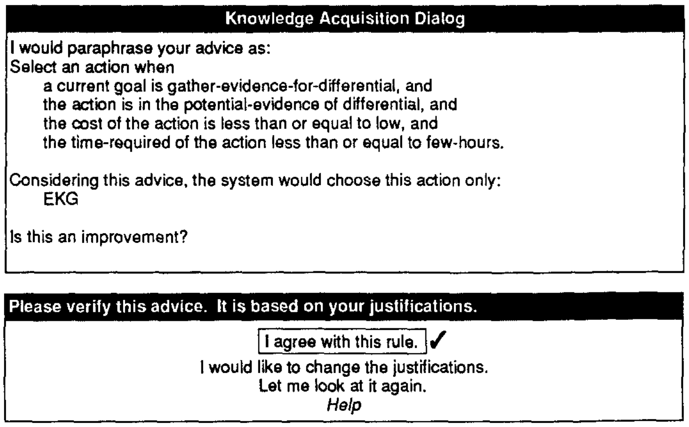
```
(= (time-required ?ACTION) few-minutes)
```

with the clause

```
(≤ (time-required ?ACTION) few-hours).
```

4.9 Verifying a Rule

To evaluate the face validity of the generated rule, ASK presents a paraphrased translation to the user for verification. It also shows the operational consequences of the rule.



I would paraphrase your advice as:
Select an action when
a current goal is gather-evidence-for-differential, and
the action is in the potential-evidence of differential, and
the cost of the action is less than or equal to low, and
the time-required of the action less than or equal to few-hours.

Considering this advice, the system would choose this action only:
EKG

Is this an improvement?

Please verify this advice. It is based on your justifications.
I agree with this rule. ✓
I would like to change the justifications.
Let me look at it again.
Help

This completes one session of the knowledge acquisition dialog. With the new strategy rule, the performance system now recommends only the positive example, EKG, when the goal is to gather evidence and the actions are potentially diagnostic. The new selection rule fails to match the negative example and the other proposed actions, and it shadows the more general rule that formerly matched all seven actions.

The next subsection demonstrates how ASK can be used to acquire tradeoffs in a utility space. It is not essential to understanding the basic approach.

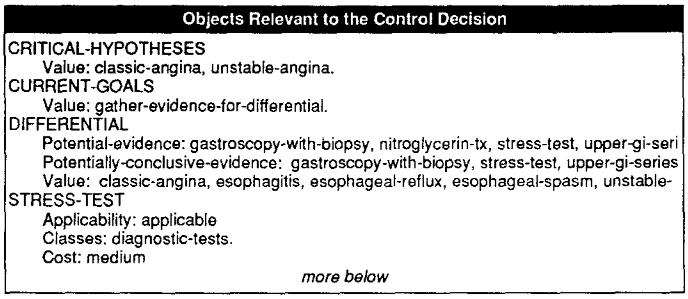
4.10. Acquiring Tradeoffs

The strategy rule just acquired is one of a family of rules that together constitute a strategy for selecting diagnostic actions. Selection rules can be viewed as tradeoffs among features, and a family of selection rules represents a set of acceptable tradeoffs. The new rule specifies that a moderate amount of time is acceptable if the cost is low and the diagnosticity is moderate.

In terms of utility theory, the new rule occupies a region in a space with dimensions defined by the features measuring diagnosticity, cost, and timeliness. Points in this space can be interpreted as the values of a multiattribute utility function [Keeney and Raiffa 1976]. The dimensions are attributes and the regions represent values of equivalent utility. The shadows relation among rules corresponds to a partial order over values of utility; some regions have higher utility than others in the same attribute space. For example, because of the shadows relation, the new rule takes precedence over selection rules that mention only cost or time-required. The region corresponding to the new rule can be interpreted as having higher utility. In other words, actions selected by the new rule are preferred over actions that would have been selected by shadowed rules.

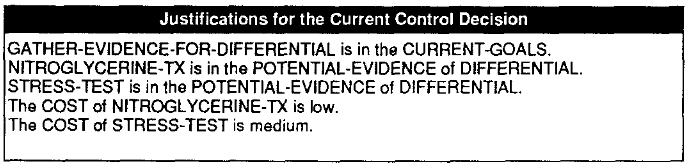
To illustrate how ASK can be used to acquire other tradeoffs in the same space, this subsection sketches a second session where the user finds an exception to an existing rule.

In this second scenario, the user runs the performance system on a case where initial data provides evidence that the patient could have a very serious condition which requires immediate diagnosis. In this situation, the system suggests a set of actions that are potential evidence for hypotheses on the differential and have low cost. However, the user indicates that the system should ignore cost and concentrate on evidence that is potentially *conclusive* for hypotheses that are *critical*. The relevant objects and justification windows appear as follows:



Objects Relevant to the Control Decision

```
CRITICAL-HYPOTHESES
    Value: classic-angina, unstable-angina.
CURRENT-GOALS
    Value: gather-evidence-for-differential.
DIFFERENTIAL
    Potential-evidence: gastroscopy-with-biopsy, nitroglycerin-tx, stress-test, upper-gi-seri
    Potentially-conclusive-evidence: gastroscopy-with-biopsy, stress-test, upper-gi-series
    Value: classic-angina, esophagitis, esophageal-reflux, esophageal-spasm, unstable-
STRESS-TEST
    Applicability: applicable
    Classes: diagnostic-tests.
    Cost: medium
                              more below
```

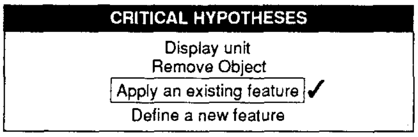


Justifications for the Current Control Decision

```
GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
NITROGLYCERINE-TX is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
STRESS-TEST is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of NITROGLYCERINE-TX is low.
The COST of STRESS-TEST is medium.
```

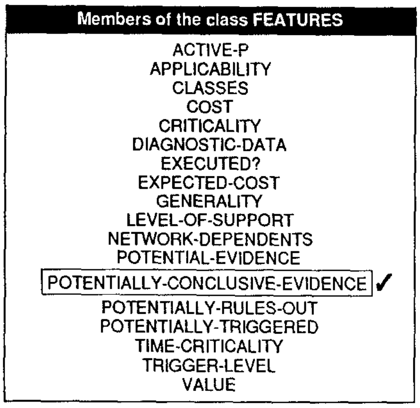
The positive example is Stress-test, which was not selected by the system because its cost was more than low. The negative example is Nitroglycerine-tx, which was selected by the system. The justifications in the window shown above were seeded by ASK; they correspond to the clauses of the strategy rules that picked Nitroglycerine-tx and not Stress-test.

In the justification session, the user tells ASK to consider *conclusive* evidence for *critical* hypotheses. The set of critical hypotheses is already represented by a knowledge-base object. Critical-hypotheses is defined as a set of hypotheses that are active (and therefore on the differential) and time-critical (a feature of hypotheses). The relationship between conclusive evidence and critical hypotheses is *not* currently represented by a feature. The relationship *is* currently defined for the set of hypotheses on the differential. Since the set of critical hypotheses and the differential share the same domain, the feature implementing the potentially-conclusive-evidence relationship can be applied to the critical-hypotheses object. The user accomplishes this by clicking on the critical-hypothesis object and performing the operations shown in the following windows.



CRITICAL HYPOTHESES

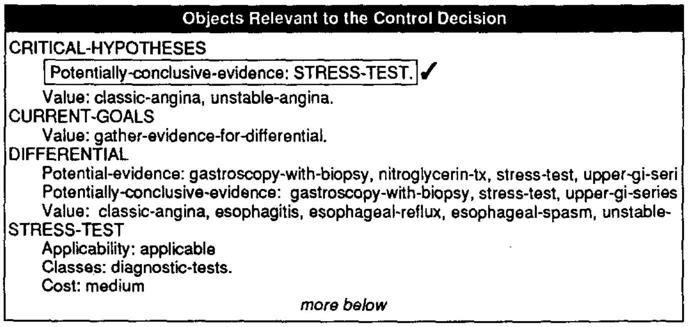
Display unit
Remove Object
[✓ Apply an existing feature]
Define a new feature



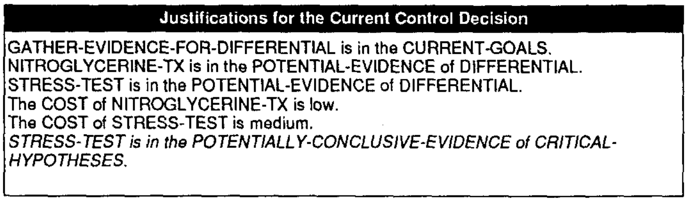
Members of the class FEATURES

ACTIVE-P
APPLICABILITY
CLASSES
COST
CRITICALITY
DIAGNOSTIC-DATA
EXECUTED?
EXPECTED-COST
GENERALITY
LEVEL-OF-SUPPORT
NETWORK-DEPENDENTS
POTENTIAL-EVIDENCE
POTENTIALLY-CONCLUSIVE-EVIDENCE ✓
POTENTIALLY-RULES-OUT
POTENTIALLY-TRIGGERED
TIME-CRITICALITY
TRIGGER-LEVEL
VALUE

The feature potentially-conclusive-evidence was conveniently defined to work for any set of hypotheses, and critical-hypotheses is a set of hypotheses. As a result, when the user applies the feature to critical-hypotheses, the set of potentially conclusive evidence for critical hypotheses is immediately computed. The newly-applied feature is displayed in the relevant objects window and becomes available as a justification. The updated relevant objects window shows the value of the feature as the singleton set containing the action Stress-test. In the window shown below the user selects this fact as a justification for choosing the stress test.



```
Objects Relevant to the Control Decision
CRITICAL-HYPOTHESES
    Potentially-conclusive-evidence: STRESS-TEST. ✓
    Value: classic-angina, unstable-angina.
CURRENT-GOALS
    Value: gather-evidence-for-differential.
DIFFERENTIAL
    Potential-evidence: gastroscopy-with-biopsy, nitroglycerin-tx, stress-test, upper-gi-seri
    Potentially-conclusive-evidence: gastroscopy-with-biopsy, stress-test, upper-gi-series
    Value: classic-angina, esophagitis, esophageal-reflux, esophageal-spasm, unstable-
STRESS-TEST
    Applicability: applicable
    Classes: diagnostic-tests.
    Cost: medium
                              more below
```



```
Justifications for the Current Control Decision
GATHER-EVIDENCE-FOR-DIFFERENTIAL is in the CURRENT-GOALS.
NITROGLYCERINE-TX is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
STRESS-TEST is in the POTENTIAL-EVIDENCE of DIFFERENTIAL.
The COST of NITROGLYCERINE-TX is low.
The COST of STRESS-TEST is medium.
STRESS-TEST is in the POTENTIALLY-CONCLUSIVE-EVIDENCE of CRITICAL-
HYPOTHESES.
```

With this set of justifications, ASK generates the rule paraphrased to the user as follows:

```
Select an action when
    a current goal is gather-evidence-for-differential, and
    the action is in the potential-evidence of differential, and
    the action is in the potentially-confirming-evidence of critical-hypotheses, and
    the cost of the action is ignored.
```

The final clause of the rule is a positive form of the *dropping conditions* generalization operator. It specifies explicitly that the cost criterion, which was mentioned in the system's existing rule, should be overridden by this new rule. The ignore clauses are used in determining the shadowing relationship among strategy rules (Section 3.5). This new rule will shadow the existing rule. The operational effect is that the actions that are potentially conclusive for critical hypotheses will be selected regardless of cost, if there are any such actions and hypotheses; otherwise, actions that provide evidence for any active hypotheses and have low cost will be selected.

5. Experience Using ASK

This section reports briefly on some test sessions performed to evaluate ASK. More detailed analysis of these experiments and the positive and negative results may be found in [Gruber 1989].

ASK has been tested for the *prospective diagnosis* task [Cohen, Greenberg, and Delisio 1987] in the domain of chest pain, which is the problem addressed by the MUM system and used as an example performance system in this article. The original MUM strategy, the strategic phase planner described in [Cohen, et al. 1987], was written by a knowledge engineer as a set of knowledge sources implemented in Lisp. ASK was used by its designer to (re)acquire MUM's strategic knowledge from scratch in the form of strategy rules.

ASK was also tested with the physician who served as the domain expert for MUM. He was able to add domain-specific strategic knowledge to an existing general strategy in dialogs like those demonstrated in Section 4. In one session, the original domain expert taught a colleague how to use ASK. In general, this experience suggested that the following conditions are important for success at helping the domain expert teach a diagnostic strategy to ASK:

- The relevant control features are defined in advance (e.g., potentially-conclusive-evidence relation of Section 4.10) or are analogous to existing features (e.g., the definition of time-required, which is analogous to cost, can be elicited by example as shown in Section 4.6). When new features have no analog, then it may require knowledge engineering skills to define them. The problem of defining features is discussed in Section 6.3.
- The user understands the opportunistic control model that underlies the strategy-rule representation. If the user does not understand how the strategic knowledge is used, he or she may not give ASK useful information upon which to build strategy rules. For example, when the second physician used ASK for the first time, he tried to get it to follow a procedure-like plan: e.g., ask *all* the history and examination questions before proposing *any* diagnostic tests. This caused ASK to construct an overly general strategy rule, as described in Section 6.3.

Some representational limitations of the strategy-rule approach to control were revealed in another experiment in which ASK was used to reimplement NEOMYCIN's diagnostic strategy. One difference between ASK's strategy rules and NEOMYCIN's tasks and metarules [Clancey 1988; Clancey and Bock 1988] is the way in which the problem-solving state is represented. In NEOMYCIN, metarules are invoked by tasks, and tasks are invoked like subroutines with arguments. Some of the problem-solving state is represented by the calling stack for task invocation. In addition, metarules access and set global variables. These computational properties make certain kinds of strategic knowledge easier to represent. The task structure serves as a natural representation for goal-directed control, and the global variables and task arguments encourage a strategy with a persistent focus on the "current hypothesis" and "current finding." In contrast, ASK's strategy rules have no hierarchical calling structure and cannot set global variables. As a consequence, it is difficult to implement a goal-directed (top-down) strategy or to manipulate the differential as a data structure. ASK's representation and the corresponding elicitation metaphor is more suited to acquiring an opportunistic strategy.

In principle, one can completely reproduce the observable behavior of the NEOMYCIN strategy using ASK, because the strategy-rule language together with MU's control features are Turing complete. In practice, knowledge engineering skills were required to coerce the desired behavior from strategy rules, mainly by defining control features. For example, the engineer using ASK had to define special control features to correspond to NEOMYCIN's "current hypothesis" and "pursued hypothesis" which were stated more naturally with metarules and variables in the NEOMYCIN language. The engineering effort went into defining sophisticated features. ASK is more helpful for building up associations between existing features and actions in strategic decisions.

6. Analysis: Scope of Applicability, Assumptions, and Limitations

Although the approach taken with ASK is independent of any domain, it necessarily sacrifices generality for power. The ASK approach commits to a method of applying strategic knowledge that iteratively chooses among individual actions, employs strategy rules for the representation, and bases new knowledge on justifications of choices of actions. As a consequence, ASK has limited scope and requires some strong assumptions. This section will characterize the scope of applicability of ASK in terms of properties of a class of performances tasks and will explicate the critical assumptions and limitations that are inherent in the approach.

6.1. Characteristics of Tasks to Which ASK Applies

The problems to which ASK can be applied are those for which expert strategy is essential to the performance task and for which the strategy-rule knowledge representation and MU architecture are adequate. This is not a circular definition; it states that the applicability of the acquisition tool depends largely on the adequacy of the performance representation. Representational adequacy is judged with respect to the class of performance tasks and the problem-solving method for which a representation is designed (see Section 2.1). So, ASK's applicability will be characterized in terms of tasks for which the strategy-rule representation and the strategy-rule control cycle are appropriate.

The major characteristics of tasks to which ASK would apply are as follows:

Actions can be selected one at a time (as opposed to sequences of action). A task for which this characteristic often holds is reactive planning for robots, where uncertainty about the world and real-time constraints necessitate acting without projection. Robots controlled by reactive planners select actions on the basis of immediate features of the environment, without projecting the consequences of several possible sequences of actions and picking the best sequence.

A task for which selecting actions one at a time is *not* appropriate is planning a set of drugs to cover an infection. MYCIN's therapy algorithm, for example, selects a collection of drugs to cover a set of infectious organisms using an algorithm written in Lisp [Clancey 1984]. This task necessitates reasoning about the *collective* properties of groups of drugs and organisms. Since the utility of individual actions depends strongly on the other actions to be selected at the same time, the strategy-rule representation could not capture the desired drug-selection expertise. (If every possible collection of drugs was represented as a "superaction," then strategy rules could represent drug-selection criteria. However, this is not feasible for large numbers of drugs, and it reduces all strategic reasoning to a single decision.)

Actions can be related directly to the situations in which they should be chosen. A positive example of a task with this property is selecting legal cases for argument, where cases are treated as actions. The merits of each case can be represented with features that describe its individual properties and its relationships to other cases and the current fact situation (e.g., Ashley's [1989] *dimensions*). A case-selection strategy might be modeled by relating the relevant features of legal situations to the features of cases that may be cited in the specified situations. For instance, in a trade secrets situation one might cite cases that make a claim about whether and how secrets were disclosed.

A negative example is the management of cancer treatment plans, the domain of the performance system ONCOCIN [Tu, et al. 1989]. The strategy for cancer treatment in ONCOCIN is represented with *protocols*: skeletal plans that are instantiated with therapeutic actions for particular patients. In attempting to model the individual treatment steps as actions in ASK, we found that the justifications for choosing the next action in cancer protocols were often statements of the form "because drug V is a member of the drug combination VAM, which is the next chemotherapy to be administered to this patient according to protocol 20-83-1" rather than "VAM is useful for small cell lung cancer because this combination can help prevent the tumor becoming resistant." The justifications also did not include a description of the context in which drugs V, A, and M are competing with other possible drugs. The knowledge underlying the recommendation of the VAM drug combination is compiled into the skeletal plan for a protocol. In this domain it is unrealistic to expect the experts to justify treatment plans with underlying reasons for their use, because by their very nature protocols are experiments designed to *test* the effectiveness of treatment strategies.

In general, the opportunistic style of control afforded by ASK's representation *generates* plans based on underlying reasons for taking plan steps (actions), when they are available. A memory-based planner unfolds and *instantiates* stored plans, in which individual actions need no independent justification. Applications requiring domain-specific strategic knowledge often do both styles of reasoning about action. *Within* an ONCOCIN protocol, for example, actions may be modified or dropped for reasons relating to the dynamic situation (e.g., the condition of the patient). Strategic knowledge for modifying steps within a plan could be formulated in strategy rules and acquired with ASK if the position of an action in a plan were abstracted as a control feature. In ONCOCIN this knowledge is, in fact, represented with rules that are indexed by protocols.[3]

Local action-selection criteria can avoid global pitfalls. Computer players of adversarial games often are based on static evaluation of position. Their strategy for selecting a next move is to choose the action that scores best on the evaluation function. If the evaluation function can be structured as a conjunctive expression over features, ASK could be used to acquire it. For example, ASK can acquire the kind of strategy learned by Waterman's poker player: mappings from descriptions of the board and the opponent to betting actions [Waterman 1970]. A game-playing strategy based on mappings from features of game situations to classes of moves will succeed if the features are usefully predictive—if what looks good locally does not lead to global pitfalls.

A borderline negative example is chess, where strategy is often played out over several moves, and evaluation functions are prone to horizon effects. If the right features can be found, strategy rules can map them to actions and ASK can acquire them. If the features invented by the user lead to pitfalls, then acquiring rules that use these features will not produce a globally optimal strategy.

In general, strategy rules support the reactive style of reasoning, where features are immediately available. In contrast, search-based planning can explore the outcomes of actions into the simulated future and back up the evaluation of the utility of the results. Therefore, ASK can be useful for tasks in which the effects of actions cannot be accurately predicted. The features acquired by ASK combine predictions of effects and the expected utility of effects.

An optimal decision among actions is not required or possible for every choice of actions. The chest pain application is both a positive and negative example. Most of the evidence-gathering questions, tests, and therapies are chosen with relatively simple measures of utility, such as qualitative measures of diagnosticity, efficacy, and cost. In practice, the data and necessity to elicit probabilities and numeric estimates of utility for every possible combination of actions is not present. However, a negative example in the same domain is the *last* strategic decision that is typically made (or avoided): deciding whether to perform angiography and consequently open heart surgery. This decision has been successfully modeled using the techniques of decision analysis [Pauker and Kassirer 1981].

There is no reason in principle why ASK's model of selecting actions cannot be described in terms of expected utility, nor is there any fundamental reason why a Bayesian utility function could not be used as a feature in strategy rules. The practical difference is in how a utility model is constructed. A set of strategy rules form a qualitative model of the utility of actions, where the union of actions recommended by rules are treated as equivalent. A multiattribute decision model [Keeney and Raiffa 1976] makes finer-grained, numeric estimates of the relative utility of each attribute, and combines them to rank the recommended actions.

6.2. Critical Assumptions

ASK makes progress in automating the acquisition of strategic knowledge, but many aspects of this difficult problem are not solved. What is left for further work is revealed by the assumptions that the ASK approach makes about the available knowledge and the people that can provide it. Some key assumptions are discussed here, and a more complete list is supplied in [Gruber 1989].

Requirements on the substantive knowledge. The ASK approach assumes that substantive knowledge of the performance system: 1) is already acquired or can be acquired, 2) is correct, and 3) is sufficient for making the distinctions necessary for the strategic knowledge.

The control features used by ASK depend on existing substantive knowledge in the inference network of a MU performance system. For example, in diagnostic tasks, much of the important substantive knowledge is found in combination functions which specify how evidential support values and other inferential values are propagated through the inference network. In the MU environment, combination functions are acquired with a symbol-level interface—editors that present and elicit knowledge in the same form as it is used (i.e., rules, slot values, etc.). ASK assumes that the MU interface is adequate for acquiring substantive knowledge.

A more serious problem is the assumption that the substantive knowledge is correct. ASK's credit assignment algorithm determines what type of rule to acquire and which objects the rule must match and not match. The algorithm is based on the assumption that the features mentioned in existing strategy rules are correct. To account for the discrepancy between system and user actions, a new rule must match different features or different values of features than the existing strategy rules. If the features return incorrect values for some actions, this algorithm cannot correctly attribute the blame.

Finally, ASK assumes that the features that are already defined or are easily defined within the existing knowledge base are sufficient for representing the desired strategy. The experiment in reimplementing NEOMYCIN described in Section 5 was an opportunity to test this assumption. NEOMYCIN's strategy makes heavy use of the subsumption relation among hypotheses. For example, one metarule specifies that, "If the hypothesis being focused upon has a child that has not been pursued, then pursue that child." The CHILD metarelation assumed by this rule is a subsumption relation among hypotheses that was not present in the MUM knowledge base used in our experiment. It was simply not possible to acquire this strategic knowledge without reorganizing the substantive knowledge base (i.e., identifying abstract categories of diseases and relating them in a hierarchy to the existing diseases).

In general, the overall effectiveness of ASK in acquiring strategic knowledge is bounded by the difficulty of representing the relevant control features for the domain.

Validity of experts' justifications for acquiring strategy. It was argued in Section 1 that the acquisition of strategic knowledge is difficult because domain experts do not normally express their strategy in a form that is generative, operational, and general (i.e., because of representation mismatch). However, it is observed that experts *can* give justifications for specific strategic decisions. The approach taken in ASK requires a strong assumption: that experts' justifications form a valid basis for acquiring the strategic knowledge of systems. There are several ways that this assumption might be wrong.

One way is the problem of tacit knowledge—that the knowledge we wish to acquire from experts is not explicitly present in what they tell us. An influential theory in cognitive science argues that the knowledge underlying expertise is often tacit due to the process of *knowledge compilation* [Anderson 1986]. As experts learn problem-solving strategies from experience in a domain, they internalize the useful associations between situations and actions and become unaware of the inferential steps that they may have made as novices. For example, physicians in an educational setting may teach diagnostic strategy one way and practice it another way. In experimental settings, when people are asked to account for their decisions retrospectively they often refer to causal theories or judgments of plausibility rather than the pertinent stimuli and their responses [Nisbett and Wilson 1977]. And some writers argue that the difference between being able to act and being able to talk about action is fundamental—that computer models of action are essentially incapable of capturing the real basis for action [Winograd and Flores 1986].

If experts cannot account for their strategic decisions, ASK cannot acquire the strategic expertise in a program. There is a difference, however, between assuming that experts can describe their own cognitive processes and assuming that they can justify their behavior. ASK only depends on the latter assumption. The assumptions that experts can provide valid justifications may be reformulated as the requirement that experts be good teachers. Remember that ASK is designed to acquire knowledge for choosing actions that are observable and, therefore, objectively justifiable. The fact that medical school professors may not practice what they preach does not mean that the justifications are invalid. On the contrary, good teachers can account for behavior in a principled way and in objective terms, even though their compiled expertise may not follow from their explanations.

A second problem with the reliance on expert-supplied justifications is the assumption that domain experts can invent useful abstractions of the domain—the right control features. In the same way that an autonomous machine-learning program is limited by the description language provided by the program author, a knowledge acquisition system such as ASK is dependent on the abstraction skills of the user.[4] ASK relies on the user to invent features that not only are sufficient to distinguish actions in specific cases, but also lay out a space of relevant generalizations. This assumption would be unfounded if the expert defined a unique feature for every training case; the resulting strategy—a lookup table of special cases—would be brittle. It is also possible that an expert can describe useful features in natural language but cannot implement them.

The validity of an assumption about the skill level of users is an empirical question, and the answers will depend on the subjects and the tasks. ASK helps frame the research question by distinguishing between the ability to *invent* the necessary features, which is structured by the elicitation of justifications, and the *implementation* of features, which is partially supported by a symbol-level interface for defining features. If an ASK user cannot implement a feature but knows what it should represent, she calls the knowledge engineer. The acquisition of features (new terms) is an interesting area for further study.

6.3. Major Limitations

Two of the fundamental limitations to the approach taken in ASK are discussed in this section. A more complete analysis is given in [Gruber 1989].

6.3.1. Reliance on Knowledge Engineering Skills It should be clear from the preceding discussion that ASK depends on the ability of the user to define and implement control features. The fact that many features are not easy to implement means that ASK is still limited by the operationalization aspect of representation mismatch. The problem of operationalizing terms is relevant to any learning system whose description language can be extended by the user. Although ASK provides a helpful interface for defining new features, some new features require programming to implement. The problem is not a matter of learning the notation; one needs to know a lot more than the syntax of Lisp to be able to implement control features that capture sophisticated assessments of the state of problem solving. To implement a feature such as the potentially conclusive-evidence relation, one needs to understand the workings of the MU architecture at the symbol level. That is the expertise of knowledge engineers, not domain experts.

There is a way in which ASK's elicitation technique can actually aggravate the problem of representation mismatch. ASK is designed to present the "user illusion" [Kay 1984] of an interface that accepts *explanations* for strategic decisions. In contrast, a symbol-level acquisition tool such as TEIRESIAS [Davis 1976] supports a straightforward interface to rules without disguising them as anything else. The problem with a system such as ASK that presents a knowledge-level interface to the user but internally makes symbol-level distinctions is that the user's model of how the system works can differ significantly from how the system actually functions. If the user's model is inaccurate, she cannot predict what the system will do with what is elicited. The result is a breakdown in communication and a failure in the knowledge acquisition process.

One of the experiments in which ASK was used by physicians illustrates a case in which the user's ignorance of the operational semantics of strategy rules resulted in an unintended strategy. The expert wanted to teach the system to ask all applicable questions of one class before asking any applicable questions of another. He answered ASK's prompts in such a way that the credit assignment algorithm determined that it needed to acquire a filter rule, when in fact a selection rule was needed. When the expert explained (with justifications) that questions of one type should *not* be selected, ASK generated a filter rule that prohibited questions of that type from *ever* being selected, which is a gross overgeneralization. The error was not apparent until the actions from the first class were exhausted and the system could not suggest any more actions to perform. To have avoided this problem, the user would have had to understand the operational difference between filter and selection rules and the correspondence between his answers to ASK's prompts and the type of rule being acquired.

6.3.2. Overgeneralization Due to the Lack of a Training Set Although ASK uses generalization operators, it differs from most inductive learning techniques in that it does not learn from a large training set of examples. The user is responsible for choosing training examples that will produce useful generalizations. Unfortunately, the lack of a large training set limits the extent to which ASK can help with the generalization problem.

It is easy to generate strategy rules with ASK that are overly general, because of the elicitation technique. Adding justifications specializes the resulting strategy rule; doing nothing leaves it general. Consider two strategic situations in the medical workup. In the early phase, actions are selected for their low cost and minimal diagnosticity. In later phases, actions that offer a potentially significant diagnostic or therapeutic value are selected at higher cost, even if lower-cost actions are available. If the selection rules for the first phase were acquired without any clauses identifying the strategic situation (i.e., features of the early phase), then the rules acquired for the early phase would also match when the later phase arose. There is no knowledge-free way for ASK to anticipate the missing clauses that specify the context in which a rule should apply.

In practice, overgeneralizations of this type are discouraged by starting with an initial set of strategy rules that specify the basic strategic situations to distinguish. These rules serve as the basis for seed justifications (Section 4.5) upon which the user builds a set of justifications for a specific case. The knowledge engineer can provide a set of very general strategy rules, anticipating some of the situations in which domain-specific tradeoffs will arise. Then the major role of the user is to *specialize* the general strategy with application-specific strategic knowledge. Overgeneralizations are still likely, however, when the user fails to elaborate the features of a novel context in which a selection is made among specific actions.

If ASK kept a library of training cases, it might be able to check newly formed rules for inconsistency with past training and prevent excessive overgeneralization. Each case in a library would need the values of all relevant features of the positive and negative examples and the features specifying the strategic situation. When a new rule is proposed, it could be tested against the objects in the case. If the new rule recommended a different outcome than the stored case, and did not shadow the rule associated with the case, then the two rules would be inconsistent. Unfortunately, keeping a library of cases is not trivial because the space of features can grow with experience. If a new rule mentions a new feature, it is incomparable with previous cases that did not mention the feature, unless the feature is static (i.e., its value does not change during the execution of the performance system). A general solution is to store a snapshot of the entire working memory with each case, so that all possible relevant features could be derived. This solution could be expensive. The whole issue of how to store experience for future learning is an intriguing avenue for research. Some promising approaches have been developed for case-based learning systems [e.g., Bareiss 1989; Hammond 1989].

7. Discussion: Key Design Decisions

Design decisions are often hidden sources of power in AI systems. This section discusses a few characteristics of ASK's design as they relate to its function as an automated knowledge acquisition tool.

The strategy-rule representation supported by ASK is neither a novel way of formulating strategy nor an ad hoc design. For the purpose of *implementing* strategic knowledge, a procedural representation such as a Lisp function or an augmented transition network would have been more flexible. The goals in designing a representation for ASK are to be able to capture strategic knowledge in an executable form *and* to be able to elicit it from experts.

Strategy rules were designed to represent mappings between states of the inference network and equivalence classes of actions, for each of three operations: propose, filter, and select. The declarative clausal form of strategy rules allows for execution by conventional unification-style matching and corresponds to the structure of justifications. Limiting the operational effects of rules to propose, filter, and select operations simplifies credit assignment and conflict resolution. The result is a representation in which strategic knowledge can be acquired.

Two of the design decisions that led to this representation are critical to ASK's techniques for automated knowledge acquisition. First, strategic knowledge has been formulated as classification knowledge. Second, a global strategy is represented as a family of strategy rules with fine-grained effects. The rationale for each decision is given below.

7.1. Formulating Strategic Knowledge as Classification Knowledge

Strategy rules structure knowledge about what to do next as knowledge for classification: associations between strategic situations and classes of actions. The following capabilities follow from this design.

The ability to use conventional machine learning techniques. ASK can use simple syntactic induction operators for generalization (turning constants into variables, dropping conditions, and extending reference). Whereas the problem of learning sequences and procedures with internal states is very hard [Dietterich and Michalski 1986], the problem of learning classification rules is well understood [Dietterich and Michalski 1983]. If mappings from states to actions define the classes of state descriptions in which actions are appropriate, a learner can generalize control knowledge by generalizing class descriptions.

The ability to elicit machine-understandable information at the knowledge level. ASK can elicit applicability conditions for control decisions in machine-understandable terms, because the justifications from the user's point of view correspond to clauses in the rule representation. The list of justifications can be elicited in any order, since they are used as conjuncts in the class descriptions.

The ability to use simple explanations for input and output. ASK can use simple template-based natural language generation to provide explanations. ASK's explanations are just lists of facts relevant to the current control decision paraphrased in English; they are essentially the same as justifications. ASK can get away with this simple explanation technique because every control decision is a flat match of situations and associated actions. Because there is no implicit state, such as there is in an evolving control plan, the context of the decision to choose an action is fully explained by the clauses of matching strategy rules. The English explanation—paraphrases of instantiated clauses—corresponds to what is happening at the symbol level.[5]

The use of explicit, abstract control knowledge for explanation was developed in the work of Swartout, et al. [Swartout 1983; Neches, Swartout, and Moore 1985] and Clancey [Clancey 1983a, 1983b]. ASK follows the principle arising from their work that an explanation of surface behavior should correspond to the structure of the system's strategy. However, in contrast to serious attempts at knowledge-system explanation, ASK's explanations do not describe the goal structure and focusing behavior of the system because the performance architecture does not support the corresponding control mechanisms (e.g., goal stacks, tasks, etc.).

The inability to acquire goal-directed plans. As a consequence of formulating strategy as simple classification, it is awkward to acquire goal-directed strategy with ASK. To capture the knowledge for reasoning about action at different abstraction levels, the strategy-rule representation would have to be extended to support hierarchical planning in the sense of ABSTRIPS [Sacerdoti 1974]. Currently, all strategy rules within each category (propose, filter, select) are matched in parallel at each iteration. In one extension proposed in [Gruber 1989], the rules would be partitioned into abstraction levels; at each level, rules would choose the subgoals for the lower abstraction level until the subgoals at the lowest level are grounded in individual actions. It is not clear whether the added structure would compromise the comprehensibility of the elicitation technique; this is a question for future research.

7.2. Formulating Strategy as Fine-Grained Reactions

Recall the third aspect of representation mismatch: domain experts have more difficulty devising a general procedure that accounts for their strategic expertise than describing what they actually do in specific cases. ASK shows that strategic knowledge can be acquired from experts if it is elicited in the context of specific choices among actions and then generalized. This is possible because strategy rules model local decisions about actions that can be generalized to classes of situations and actions. In theory, what appears to be a global strategy can emerge from a series of local strategic decisions. For example, Chapman and Agre [1986] propose that complex, coherent behavior arises from the continued activation of situation-action structures without top-down control.

There is empirical support for the notion that globally coherent plans can be acquired by eliciting the knowledge for local decisions. For example, SALT succeeds at acquiring knowledge about how to construct globally satisfactory solutions to a class of design problems [Marcus 1987, 1988]. SALT elicits from designers knowledge about constraints among individual parts—information that is relatively easy to specify—and offers help for putting the pieces together. SALT's results are relevant to ASK because constructing a solution requires managing the *process* by which parts are assembled under constraints; this is similar to managing the selection of actions. SALT can acquire the requisite knowledge from experts because it decomposes the larger task of assembling a solution into small decisions about what part to add, how to (immediately) check it for constraints, and how to recover from those violated constraints.

One can view SALT's design task and ASK's action-selection task as varieties of planning, where configured parts and diagnostic actions correspond to plan steps. This view reveals an important difference between the two architectures. SALT's planning method provides for a backtracking search, whereas ASK's planning method is purely reactive, with no projection (lookahead) and no possibility to undo actions. This may prove to be an important variable in the question of whether knowledge of local decisions can add up to a global strategy.

8. Conclusion

The immediate outcome of this research is a method for partially automating the acquisition of strategic knowledge from experts. The issues that are raised, however, are more significant than the ASK program itself. Strategic knowledge was chosen for the study of knowledge acquisition because it illuminates the problems of representation mismatch. Furthermore, an extreme solution was selected—a declarative representation of reactive control knowledge—to test conjectures about sources of power for knowledge acquisition. The results have been analyzed in the preceding discussions of the scope of applicability, assumptions, limitations, and design decisions. This section concludes with a more general point brought out by this work and the future research it suggests.

If representation mismatch describes the problem of knowledge acquisition, then solutions should offer some way to bridge the representational gap between the domain expert and the implementation. This suggests that the design of knowledge representations is central to addressing the knowledge acquisition problem. This article has emphasized the motivations for and implications of ASK's representation of strategic knowledge in an effort to elucidate principles of *design for acquisition*: how to design knowledge systems to facilitate the acquisition of the knowledge they need.

Earlier reports [Bylander and Chandrasekaran 1987; Gruber and Cohen 1987] describe how knowledge representations and methods for task-level architectures can facilitate *manual* knowledge acquisition (i.e., mediated by tools that are passive). The design of representations can reduce representation mismatch from the implementation side by providing (generic) task-level primitives which enable experts to work directly with the knowledge base.

The ASK research illustrates how *automated* knowledge acquisition can help overcome representation mismatch by eliciting knowledge in a form that is available from experts and yet is very close to an operational, generalizable representation. Again, the design of representations plays a central role in the success of the knowledge acquisition process. The major contributions of ASK to the process—active elicitation of justifications, credit assignment, and syntactic generalization—are enabled by the declarative, role-restricted rule representation. At the same time, the kind of strategic knowledge that can be acquired—opportunistic and reactive rather than goal-directed and plan-driven—is a function of what can be naturally represented in strategy rules.

A similar power/generality tradeoff can be found in most knowledge acquisition tools. At the power end of the continuum lie OPAL-class elicitation tools [Freiling and Alexander 1984; Gale 1987; Musen, et al. 1987], which acquire knowledge in representations customized to a problem-solving method and a particular domain. OPAL employs elicitation techniques that are customized for both the skeletal-plan refinement method used in ONCOCIN and the domain of cancer therapy. As a result, OPAL can be used by domain experts. At the generality end lie TEIRESIAS-class tools [Davis 1976; Boose and Bradshaw 1987; Shachter and Heckerman 1987], which acquire knowledge at the symbol-level for formalisms that are not committed to particular tasks or domains. TEIRESIAS makes it easy to enter and modify rules but requires the user to bridge the representational gap from the domain- and problem-specific description to the backward-chaining architecture. Somewhere in the middle are the MOLE-class tools [Eshelman 1988; Klinker 1988; Marcus 1988], which acquire knowledge in representations that are method-specific and domain-independent. This article has shown several ways in which the design of ASK trades the generality of a representation useful for knowledge engineering for the power of a restricted representation suitable for automated knowledge acquisition.

Further research is needed to investigate how knowledge representations and reasoning methods can be designed to make the task of knowledge acquisition more amenable to computer-assisted techniques for elicitation and learning.

Acknowledgments

This article reports on the author's doctoral research directed by Paul Cohen at the University of Massachusetts, Department of Computer and Information Science. The work was supported by DARPA-RADC contract F30602-85-C0014 and ONR University Research Initiative contract N00014-86-K-1764. The paper was written at the Stanford Knowledge Systems Laboratory with funding from Tektronics, Inc. Computing facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785.

Discussions with many colleagues have contributed to the ideas in this paper, including Kevin Ashley, Jim Bennett, B. Chandrasekaran, Bill Clancey, David Day, Larry Fagan, Richard Fikes, Michael Freiling, Victor Lesser, John McDermott, Mark Musen, Paul Utgoff, and especially Paul Cohen. I am very grateful to Ray Bareiss, Tilda Brown, Paul Cohen, Richard Keller, Sandra Marcus, Mark Musen, Bruce Porter, and Nancy Wogrin for thoughtful and careful reviews of earlier drafts of this paper.

Notes

1. Dietterich and Bennett [1988] refer to "making goals achievable" and "making goals more useful."
2. Control features correspond to the *metarelations* in Clancey's tasks-and-metarules representation [Clancey and Bock 1988].
3. Thanks to Lawrence Fagan, Mark Musen, and Samson Tu for their help with this analysis.
4. Getting the right primitive features has always been essential to getting a machine learning program to find useful generalizations. For example, Quinlan [1983] reports having spent three months devising a good set of attributes (board position features for chess) so that the learning program ID3 could produce a decision tree in seconds.
5. This is an oversimplification. In actuality, the shadowing relations among strategy rules are not reflected in the explanation. Not surprisingly, they are a source of confusion for users, possibly because they do not fit the simple conceptual model of situation–action.

References

Agre, P.E., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 268–272). Seattle, Washington: Morgan Kaufmann.

Anderson, J.R. 1986. Knowledge compilation: the general learning mechanism. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*, (Vol. 2). San Mateo, CA: Morgan Kaufmann.

Ashley, K.D. 1989. *Modelling legal argument: Reasoning with cases and hypotheticals*. Cambridge, MA: MIT Press. Based on doctoral dissertation, Department of Computer and Information Science, University of Massachusetts, Amherst.

Bareiss, E.R. 1989. *Exemplar-based knowledge acquisition: A unified approach to concept representation, classification, and learning*. Boston: Academic Press. Based on doctoral dissertation, Department of Computer Science, University of Texas, Austin.

Benjamin, D.P. 1987. Learning strategies by reasoning about rules. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 256–259). Milan, Italy: Morgan Kaufmann.

Bennett, J.S. 1985. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning, 1*, 49–74.

Boose, J.H. 1986. *Expertise Transfer for Expert System Design*. New York: Elsevier.

Boose, J.H., and Bradshaw, J.M. 1987. Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for expert systems. *International Journal of Man-Machine Studies, 26*, 21–25.

Buchanan, B.G., Barstow, D.K., Bechtel, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D.A. 1983. Constructing an expert system. In F. Hayes-Roth, D.A. Waterman, and D.B. Lenat (Eds.), *Building expert systems*. Reading, MA: Addison-Wesley.

Buchanan, B.G., and Shortliffe, E.H. 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley.

Bylander, R., and Chandrasekaran, B. 1987. Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies, 26*, 231–244.

Chandrasekaran, B. 1983. Toward a taxonomy of problem-solving types. *AI Magazine, 4*, 9–17.

Chandrasekaran, B. 1986. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert, 1*, 23–30.

Chandrasekaran, B. 1987. Towards a functional architecture for intelligence based on generic information processing tasks. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 1183–1192). Milan, Italy: Morgan Kaufmann.

Chapman, D., and Agre, P.E. 1987. Abstract reasoning as emergent from concrete activity. In M.P. Georgeff and A.L. Lansky (Eds.), *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon* (pp. 411–424).

Clancey, W.J. 1983a. The epistemology of a rule-based expert system—A framework for explanation. *Artificial Intelligence, 20*, 215–251.

Clancey, W.J. 1983b. The advantages of abstract control knowledge in expert system design. *Proceedings of the Third National Conference on Artificial Intelligence* (pp. 74–78). Washington, D.C.: Morgan Kaufmann.

Clancey, W.J. 1984. Details of the revised therapy algorithm. In B.G. Buchanan and E.H. Shortliffe (Eds.), *Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Reading, MA: Addison-Wesley.

Clancey, W.J. 1985. Heuristic classification. *Artificial Intelligence, 27*, 289–350.

Clancey, W.J. 1988. Acquiring, representing, and evaluating a competence model of diagnosis. In Chi, Glaser, and Farr (Eds.), *Contributions to the nature of expertise* (pp. 343–418). Hillsdale, N.J.: Lawrence Erlbaum. Previously published as KSL Memo 84-2, Stanford University, February, 1984.

Clancey, W.J. 1989. Viewing knowledge bases as qualitative models. *IEEE Expert, 4*, 9–23. Previously published as Technical Report KSL-86-27, Stanford University.

Clancey, W.J., and Bock, C. 1988. Representing control knowledge as abstract tasks and metarules. In L. Bolc and M. Coombs (Eds.), *Expert system applications* (pp. 1-77). New York: Springer-Verlag.

Cohen, P.R., Day, D.S., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., and Berman, P. 1987. Management of uncertainty in medicine. *International Journal of Approximate Reasoning, 1*, 103–116.

Cohen, P.R., Greenberg, M., and Delisio, J. 1987. MU: A development environment for prospective reasoning systems. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 783–788). Seattle, Washington: Morgan Kaufmann.

Davis, R. 1976. *Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases*. Doctoral dissertation, Computer Science Department, Stanford University. Reprinted in R. Davis and D.B. Lenat (Eds.), *Knowledge-based systems in artificial intelligence*. New York: McGraw-Hill, 1982.

DeJong, G., and Mooney, R.J. 1986. Explanation-based learning: An alternative view. *Machine Learning*, *1*, 145–176.

Dietterich, T.G., and Bennett, J.S. 1988. Varieties of operationality. (Technical Report). Department of Computer Science, Oregon State University.

Dietterich. T.G., London, B., Clarkson, K., and Dromey, G. 1982. Learning and inductive inference. In P.R. Cohen and E. Feigenbaum (Eds.), *The handbook of artificial intelligence (Vol. 3)*. Menlo Park, CA: Addison-Wesley.

Dietterich, T.G., and Michalski, R.S. 1983. A comparative review of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach*. San Mateo, CA: Morgan Kaufmann.

Dietterich, T.G., and Michalski, R.S. 1986. Learning to predict sequences. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach (Vol. 2)*. San Mateo, CA: Morgan Kaufmann.

Erman, L.D., Scott, A.C., and London, P.E. 1984. Separating and integrating control in a rule-based tool. *Proceedings of the IEEE Workshop of Knowledge-base Systems* (pp. 37–43). Denver, Colorado.

Eshelman, L. 1988. MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems*. Boston: Kluwer Academic Publishers.

Firby, R.J. 1987. An investigation into reactive planning in complex domains. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 202–206). Seattle, Washington: Morgan Kaufmann.

Freiling, M.J., and Alexander, J.H. 1984. Diagrams and grammars: Tools for mass producing expert systems. *Proceedings of the First Conference on Artificial Intelligence Applications* (pp. 537–543). Denver, Colorado: IEEE Computer Society Press.

Friedland, P.E., and Iwasaki, Y. 1985. The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, *1*, 161–208.

Gale, W.A. 1987. Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man-Machine Studies*, *13*, 81–116.

Golding, A., Rosenbloom, P.S., and Laird, J.E. 1987. Learning general search control from outside guidance. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 334–337). Milan, Italy: Morgan Kaufmann.

Gruber, T.R. 1989. *The Acquisition of Strategic Knowledge*. Boston: Academic Press. Based on doctoral dissertation, Department of Computer and Information Science, University of Massachusetts.

Gruber, T.R., and Cohen, P.R. 1987. Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *International Journal of Man-Machine Studies*, *26*, 143–159.

Hammond, K.J. 1989. *Case-based Planning: Viewing Planning as a Memory Task*. Boston: Academic Press. Based on doctoral dissertation, Computer Science Department, Yale University.

Hannan, J., and Politakis, P. 1985. ESSA: An approach to acquiring decision rules for diagnostic expert systems. *Proceedings of the Second Conference on Artificial Intelligence Applications* (pp. 520–525). Orlando, Florida: IEEE Computer Society Press.

Hayes-Roth, B. 1985. A blackboard architecture for control. *Artificial Intelligence*, *26*, 251–321.

Hayes-Roth, B., Garvey, A., Johnson, M., and Hewett, M. 1987. *A layered environment for reasoning about action*. (Technical Report KSL 86-38). Stanford, CA: Computer Science Department, Stanford University.

Hayes-Roth, B., and Hewett, M. 1985. *Learning control heuristics in a blackboard environment*. (Technical Report HPP-85-2). Stanford, CA: Computer Science Department, Stanford University.

Hutchins, E.L., Hollan, J.D., and Norman, D.A. 1986. Direct manipulation interfaces. In D.A. Norman, and S.W. Draper (Eds.), *User centered system design*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Johnson, N.E., and Tomlinson, C.M. 1988. Knowledge representation for knowledge elicitation. *Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, November. Calgary, Alberta: SRDG Publications, Department of Computer Science, University of Calgary.

Kaelbling, L.P. 1987. An architecture for intelligent reactive systems. In M.P. Georgeff and A.L. Lansky (Eds.), *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon* (pp. 411–424). Morgan Kaufmann.

Kassirer, J.P., and Gorry, G.A. 1978. Clinical problem solving: A behavioral analysis. *Annals of Internal Medicine, 89,* 245–255.

Kay, A. 1984. Computer software. *Scientific American, 251,* 52–59, September.

Keeney, R.L., and Raiffa, H. 1976. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* John Wiley and Sons.

Keller, R.M. 1988. Defining operationality for explanation-based learning. *Artificial Intelligence, 35,* 227–241.

Klinker, G. 1988. KNACK: Sample-driven knowledge acquisition for reporting systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems.* Boston: Kluwer Academic Publishers.

Laird, J.D., Newell, A., and Rosenbloom, P.S. 1987. SOAR: An architecture for general intelligence. *Artificial Intelligence, 33,* 1–64.

Lenat, D.B., and Brown, J.S. 1984. Why AM and EURISKO appear to work. *Artificial Intelligence, 23,* 269–294.

Marcus, S. 1987. Taking backtracking with a grain of SALT. *International Journal of Man-Machine Studies, 24,* 383–398.

Marcus, S. 1988. SALT: A knowledge acquisition tool for propose-and-refine systems. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems.* Boston: Kluwer Academic Publishers.

McDermott, J. 1988. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating knowledge acquisition for expert systems.* Boston: Kluwer Academic Publishers.

Minton, S., and Carbonell, J.G. 1987. Strategies for learning search control rules: An explanation-based approach. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Milan, Italy: Morgan Kaufmann.

Mitchell, T.M. 1982. Generalization as search. *Artificial Intelligence, 18,* 203–226.

Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T. 1986. Explanation-based generalization: A unifying view. *Machine Learning, 1,* 56–80.

Mitchell, T.M., Mahadevan, S., and Steinberg, L.I. LEAP: A learning apprentice for VLSI design. *Proceedings of the Ninth International Conference on Artificial Intelligence* (pp. 573–580). Los Angeles, CA: Morgan Kaufmann.

Mitchell, T.M., Utgoff, P.E., and Banerji, R.B. 1983. Learning by experimentation: Acquiring and refining problem-solving heuristics. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach,* San Mateo, CA: Morgan Kaufmann.

Mostow, D.J. 1983. Machine transformation of advice into a heuristic search procedure. In R. Michalski, J. Carbonell, and T.M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Morik, K. 1988. Sloppy modeling. In K. Morik (Ed.), *Knowledge representation and organization in machine learning.* Berlin: Springer-Verlag, in press.

Musen, M.A. 1989. *Automated Generation of Model-based Knowledge-Acquisition Tools,* London: Pitman. Based on doctoral dissertation, Computer Science Department, Stanford University.

Musen, M.A., Fagan, L.M., and Shortliffe, E.H. 1986. Graphical specification of procedural knowledge for an expert system. *Proceedings of the 1986 IEEE Computer Society Workshop in Visual Languages* (pp. 167–178). Dallas, Texas.

Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H. 1987. Use of a domain model to drive an interactive knowledge editing tool. *International Journal of Man-Machine Studies, 26,* 105.

Neches, R., Swartout, W., and Moore, J. 1985. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering, SE-11* (11), 1337–1351.

Newell, A. 1982. The knowledge level. *Artificial Intelligence, 18,* 87–127.

Nisbett, R., and Wilson, T. 1977. Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84,* 231–259.

Pauker, S.G., and Kassirer, J.P. 1981. Clinical decision analysis by personal computer. *Archives of Internal Medicine, 141,* 1831–1837.

Quinlan, J.R. 1983. Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine learning: An artificial intelligence approach.* San Mateo, CA: Morgan Kaufmann.

Quinlan, J.R. 1986. Induction of decision trees. *Machine Learning, 1* (1), 81–106.

Sacerdoti, E.D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence, 5,* 115–135.

Shachter, R.D., and Heckerman, D.E. 1987. Thinking backward for knowledge acquisition. *AI Magazine 8,* 55–61.

Shaw, M.L.G., and Gaines, B. 1987. Techniques for knowledge acquisition and transfer. *International Journal of Man-Machine Studies, 27.*

Shortliffe, E.H., Scott, A.C., Bischoff, M.B., van Melle, W., and Jacobs, C.D. 1981. ONCOCIN: An expert system for oncology protocol management. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 876–881). Vancouver, British Columbia: Morgan Kaufmann.

Silver, B. 1986. *Metal-level Inference: Representing and Learning Control Information in Artificial Intelligence.* New York: North-Holland.

Sticklen, J., Chandrasekaran, B., and Josephson, J.R. 1985. Control issues in classificatory diagnosis. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence* (pp. 300–306). Los Angeles, CA: Morgan Kaufmann.

Swartout, W. 1983. XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence, 11,* 115–144.

Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H., and Fagan, L.M. 1989. Episodic monitoring of time-oriented data for heuristic skeletal-plan refinement. *Communications of the ACM,* in press.

Utgoff, P. 1986. *Machine Learning of Inductive Bias.* Boston: Kluwer Academic Publishers. Based on doctoral dissertation, Department of Computer Science, Rutgers University.

Waterman, D.A. 1970. Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence, 1,* 121–170.

Winograd, T., and Flores, F. 1987. *Understanding Computers and Cognition.* Reading, MA: Addison-Wesley.

Winston, P.H. 1985. Learning structural descriptions from examples. In P.H. Winston (Ed.), *The Psychology of Computer Vision.* New York: McGraw Hill.